New State Recovering Attack on RC4 (Full Version)

Alexander Maximov and Dmitry Khovratovich

Laboratory of Algorithmics, Cryptology and Security
University of Luxembourg
6, rue Richard Coudenhove-Kalergi, L-1359 Luxembourg
Alexander.Maximov@ericsson.com, khovratovich@gmail.com

Abstract. [1] The stream cipher RC4 was designed by R. Rivest in 1987, and it has a very simple and elegant structure. It is probably the most deployed cipher on the Earth.

In this paper we analyse the class RC4-N of RC4-like stream ciphers, where N is the modulus of operations, as well as the length of internal arrays. Our new attack is a state recovering attack which accepts the keystream of a certain length, and recovers the internal state. For the original RC4-256, our attack has the total complexity around 2^{241} of operations, whereas the best previous attack needs 2^{779} of time. Moreover, we show that if the secret key is of length N bits or longer, the new attack works faster than an exhaustive search. The algorithm of the attack was implemented and verified on small cases.

Keywords: RC4, state recovering attack, key recovering attack.

1 Introduction

RC4 [Sma03] is a stream cipher designed by Ron Rivest in 1987, since when it was implemented in many various software applications to ensure privacy of communication. It is, perhaps, the most widely deployed stream cipher on the Earth, its most common application is to protect Internet traffic in the SSL protocol. Moreover, it has been implemented in Microsoft Lotus, Oracle Secure SQL, etc. The design of RC4 was kept secret until 1994 when it was anonymously leaked to the members of Cypherpunk community. A bit later the correctness of the algorithm was confirmed.

In this paper we study a family RC4-N of RC4 like stream ciphers, where N is the modulus of operations. The internal state of RC4 is two registers $i, j \in \mathbb{Z}_N$ and a permutation S of all elements of $\mathbb{Z}_N$. Thus, RC4 has a huge state of $\log_2(N^2 N!)$ bits. For the original version, when $N = 256$, the size of the state is ≈ 1700 bits. This makes any time-memory trade-off attack impractical. RC4-256 uses a variable length key from 1 to 256 bytes for its initialisation.

The initialisation procedure of RC4 has been thorougly analysed in a large number of various papers, see e.g. [MS01,Man01,PP04]. These results show that

[1] We thank anonimous reviewers of EUROCRYPT'08 for their editorial comments.

the initialisation of RC4 is weak, and the secret key can be recovered with a small portion of data/time. Because of these attacks, RC4 can be regarded as broken. However, if one would tweak the initialisation procedure, the cipher becomes secure again.

The simplicity of the keystream generating algorithm of RC4 attracts a huge attention to its analysis. In the most of such analyses the scenario assumes that the keystream of some length is given, and either a distinguishing ([Gol97,FM00,Max05,Man05]) or a state recovering ([KMP+98]) attack is the one of the interest. A *state recovering attack* determines the actual security level of a cipher, if the initial internal state is considered as a secret key. The first such an attack was proposed by Knudsen et al in 1998 in [KMP+98], the complexity of which was 2^{779}. Some minor improvements were found in other literature ([MT98]), but still, there is no attack even close to 2^{700}. One interesting attempt to improve the analysis was done in [Man05]. Although that attack does not actually work, the pretending time complexity claimed was around 2^{290}.

In this paper we propose a new state recovering attack on RC4-N. For the original design RC4-256 the total time complexity of the attack is less than 2^{241}, and it requires the keystream of a similar size. This means that the secret key cannot be longer than 30 bytes. We also show that in general required time is less than the one an exhaustive search needs, if the secret key is of length N bits or longer.

The idea of the new attack is as follows. The algorithm searches for a place in the keystream where the probability of a specific internal state, compliant with a chosen pattern, is high. Afterwards, the new state recovering algorithm needs only a small portion of data (around $2N$ output words) in order to recover the internal state of the cipher in an iterative manner. This algorithm was implemented and verified for **small values of** N. The state recovering attack was successful and revealed a correct internal state on *every run* of the simulations. The success rate of the full attack is shown to be at least 98%. For **large values of** N, where real attack simulations were impossible, an upper bound for average complexity of the attack was derived and calculated.

This paper is organized as follows. In Section 2 new iterative *state recovering algorithm* is described in detail. Afterwards, Section 3 introduces various properties of a pattern that are needed for the recovering algorithm, and an effective searching algorithm to find such patterns is also proposed. Section 4 describes several techniques to detect specific states by observing the keystream, and also introduces additional properties of a pattern needed for detection purposes. Theoretical analysis of the state recovering algorithm and derivation of its complexity functions are performed in Appendix A (due to the page limitation). All pieces of the attack are combined in Section 5. Finally, we perform a set of simulations of the attack, summarize the results and conclude in Section 6. The paper ends with suggestions for further improvements and open problems in Section 7.

1.1 Notations

All internal variables of RC4 are over the ring $\mathbb{Z}_N$, where N is the size of the ring. To specify a particular instance of the cipher we denote it by RC4-N. Thus, the original design is RC4-256. Whenever applicable, $+$ and $-$ are performed in modulo N. At any time t the notation a_t denotes the value of a variable a at time t. The keystream is denoted by $\mathbf{z} = (z_1, z_2, \ldots)$. In all tables probabilities and complexities will be given in a logarithmical form base 2.

1.2 Description of the Keystream Generator RC4-N

We skip the description of the initialisation process since it is not in the focus of this paper. However, the full description of RC4 can be found in, e.g., [Sma03]. After the initialisation procedure, the keystream generation algorithm of RC4 begins. Its description is given in Figure 1.

Internal variables:
$i,\ j$ – integers in $\mathbb{Z}_N$
$S[0\ldots N-1]$ – a permutation of integers $0\ldots N-1$
The keystream generator RC4-N
1. $S[\cdot]$ is initialised with the secret key
   $i = j = 0$
2. Loop until we get enough symbols over $\mathbb{Z}_N$
   (A) $i = i + 1$
   (B) $j = j + S[i]$
   (C) `swap`$(S[i], S[j])$
   (D) $z_t = S[S[i] + S[j]]$

Fig. 1. The keystream generation algorithm of RC4-N.

2 New State Recovering Algorithm

2.1 Previous Analysis: Knudsen's Attack

In [KMP+98] Knudsen et al. have presented a basic recursive algorithm to recover the internal state of RC4. It starts at some point t in the keystream $\mathbf{z}$ given k known cells of the permutation S_t, which helps the recursion to cancel unlikely branches. The idea of the algorithm is simple. At every time t we have four unknowns:

$$j_t,\ S_t[i_t],\ S_t[j_t],\ S_t^{-1}[z_t]. \tag{1}$$

One can simply simulate the PRGA and, when necessary, guess these unknown values in order to continue the simulation. The recursion steps backward when a contradiction is reached, due to the previously wrong guesses. Additionally, it can be assumed that some k values are apriori known (guessed, given, or derived somehow), and this may reduce the complexity of the attack significantly. An important note is that the known k values should be located in a short window of the "working area" of the keystream, otherwise they cannot help to cancel hopeless branches.

The precise complexity of the attack was calculated in [KMP+98], and several tables for various values of N and k were given in Appendices D.1 and D.2 in [Man01]. As an example, the complete state recovering attack on RC4-256 would require time around 2^{779}.

2.2 Our Algorithm in Brief

In this section we propose an improved version of the state recovering algorithm. Assume at some time t in a window of length $w+1$ of the keystream $\mathbf{z}$ all the values $j_t, j_{t+1}, j_{t+2}, \ldots, j_{t+w}$ are known. This means that for w steps the values $S_{t+1}[i_{t+1}], \ldots, S_{i+w}[i_{t+w}]$ are known as well, since they are derived as

$$S_{t+1}[i_{t+1}] = j_{t+1} - j_t, \quad \forall t. \tag{2}$$

Consequently, w equations of the following kind can be collected:

$$S_k^{-1}[z_k] = S_k[i_k] + S_k[j_k], \quad k = t+1, \ldots, t+w, \tag{3}$$

where only *two* variables are unknown

$$S_k^{-1}[z_k], \quad S_k[j_k], \tag{4}$$

instead of *four* in Knudsen's attack, see (1). Let the set of consecutive w equations of the form (3) be called *a window of length w*.

Since all js in the window are known, then all swaps done during these w steps are known as well. This makes it possible to map the positions of the internal state S_t at any time t to the positions of some chosen *ground state* S_{t_0} at some ground time t_0 in the window. Let us for simplicity set $t_0 = 0$.

Our new state recovering algorithm is a recursive algorithm, and it is shown in Figure 2. It starts with a collection of w equations, and attempts to solve them. A single equation is called *solved* or *processed* if its corresponding unknowns (4) have been explicitly derived or guessed. During the process, the window will dynamically increase and decrease. When the length of the window w is long enough (say, $w = 2N$), and all equations are solved, the ground state S_0 is likely to be fully recovered.

A more detailed description of the parts of the algorithm follows.

Iterative Recovering (IR) Block The *Iterative Recovering* block receives as an input a number a of *active* equations (not yet processed) in the window of length w, and tries to derive the values of $S_t[j_t]$s and $S_t^{-1}[z_t]$s. For this purpose,

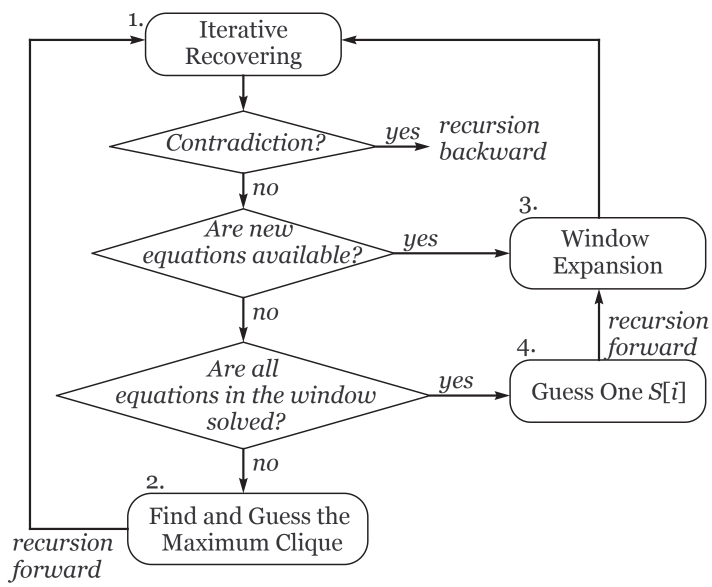

Fig. 2. New state recovering algorithm.

the IR block iteratively performs two steps, until there are no more new derivations possible. If all previous guesses were correct, then all newly derived values (cells of the ground state) will be correct with probability 1. Otherwise, when the IR block catches a contradiction the recursion makes a backward step. These two steps are as follows.

A. Assume for one of the active equations its output symbol z_t is already allocated somewhere in the ground state. I.e., the value $S_t^{-1}[z_t]$ is known, and the second unknown $S_t[j_t]$ can explicitly be derived via (3).
A contradiction is received if (a) $S_t[j_t]$ is already allocated and it is not equal to the derived value; (b) the derived value already exists at some other cell.
B. Just allocated values may give the value of $S_t[j_t]$ in another equation. Consequently, a new value $S_t^{-1}[z_t]$ can be derived via (3), which might possibly cause a contradiction.

Figure 3 illustrates the process of the IR block. In that example we start with specific values of i and j, and also $d = 5$ cells of the state S are filled with certain values, whereas the remaining cells are unknown. This constraint allows to collect $w = 15$ equations of the form (3). The keystream is given in the most right column of the table.

The first iteration, in Figure 3(b), finds that $z_6 = 4$ and $z_8 = -2$ are already allocated, thus solving equations 6 and 8 ($s_4 = 10, s_9 = 5$). Afterwards, given $s_9 = 5$, the IR block solves the equation 14 and successfully checks for a contradiction, in Figure 3(c). Finally, after the step (e) four additional cells of the state S were derived with probability 1.

		The part of the state S_t at time t, just before the swap-operation																						
i_{t+1}	j_{t+1}	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	$S[i]$	$S[j]$	z
1	8	4	-2	1	8	-4	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}	s_{12}	s_{13}	s_{14}	s_{15}	4	s_3	18
2	6	s_3	-2	1	8	-4	s_1	s_2	4	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}	s_{12}	s_{13}	s_{14}	s_{15}	-2	s_1	29
3	7	s_3	s_1	1	8	-4	-2	s_2	4	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}	s_{12}	s_{13}	s_{14}	s_{15}	1	s_2	6
4	15	s_3	s_1	s_2	8	-4	-2	1	4	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}	s_{12}	s_{13}	s_{14}	s_{15}	8	s_{10}	16
5	11	s_3	s_1	s_2	s_{10}	-4	-2	1	4	s_4	s_5	s_6	s_7	s_8	s_9	8	s_{11}	s_{12}	s_{13}	s_{14}	s_{15}	-4	s_6	5
6	9	s_3	s_1	s_2	s_{10}	s_6	-2	1	4	s_4	s_5	-4	s_7	s_8	s_9	8	s_{11}	s_{12}	s_{13}	s_{14}	s_{15}	-2	s_4	4
7	10	s_3	s_1	s_2	s_{10}	s_6	s_4	1	4	-2	s_5	-4	s_7	s_8	s_9	8	s_{11}	s_{12}	s_{13}	s_{14}	s_{15}	1	s_5	12
8	14	s_3	s_1	s_2	s_{10}	s_6	s_4	s_5	4	-2	1	-4	s_7	s_8	s_9	8	s_{11}	s_{12}	s_{13}	s_{14}	s_{15}	4	s_9	-2
9	12	s_3	s_1	s_2	s_{10}	s_6	s_4	s_5	s_9	-2	1	-4	s_7	s_8	4	8	s_{11}	s_{12}	s_{13}	s_{14}	s_{15}	-2	s_7	21
10	13	s_3	s_1	s_2	s_{10}	s_6	s_4	s_5	s_9	s_7	1	-4	-2	s_8	4	8	s_{11}	s_{12}	s_{13}	s_{14}	s_{15}	1	s_8	6
11	9	s_3	s_1	s_2	s_{10}	s_6	s_4	s_5	s_9	s_7	s_8	-4	-2	1	4	8	s_{11}	s_{12}	s_{13}	s_{14}	s_{15}	-4	s_7	9
12	7	s_3	s_1	s_2	s_{10}	s_6	s_4	s_5	s_9	-4	s_8	s_7	-2	1	4	8	s_{11}	s_{12}	s_{13}	s_{14}	s_{15}	-2	s_5	1
13	8	s_3	s_1	s_2	s_{10}	s_6	s_4	-2	s_9	-4	s_8	s_7	s_5	1	4	8	s_{11}	s_{12}	s_{13}	s_{14}	s_{15}	1	s_9	10
14	12	s_3	s_1	s_2	s_{10}	s_6	s_4	-2	1	-4	s_8	s_7	s_5	s_9	4	8	s_{11}	s_{12}	s_{13}	s_{14}	s_{15}	4	s_5	16
15	20	s_3	s_1	s_2	s_{10}	s_6	s_4	-2	1	-4	s_8	s_7	4	s_9	s_5	8	s_{11}	s_{12}	s_{13}	s_{14}	s_{15}	8	s_{15}	17
16	?																							

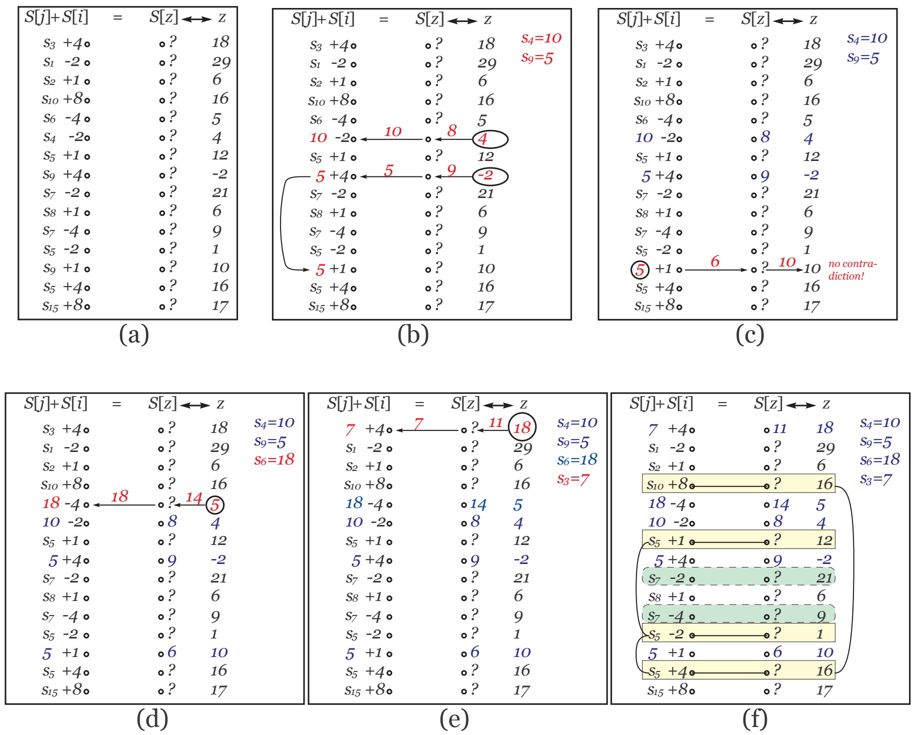

Fig. 3. Example of the iterative reconstruction process.

Find and Guess the Maximum Clique (MC) Block If no more active equations can explicitely be solved, one of $S_t^{-1}[z_t]$'s has to be guessed. The

Find and Guess the Maximum Clique block analyses given active equations and chooses such an element to be guessed that gives the maximum number of new derivations in consecutive recursive calls of the IR block.

A simple analysis is applied. Let a active equations in a graph representation be vertices v_ts. Two vertices $v_{t'}$ and $v_{t''}$ are connected if $z_{t'} = z_{t''}$ or/and $S_{t'}[j_{t'}]$ and $S_{t''}[j_{t''}]$ *refer* (like pointers) to the same cell of the ground state. A guess of any unknown variable in any connected subgraph solves all equations involved in that subgraph. Therefore, let us call these subgraphs by *cliques*. The MC block searches for a maximum clique, and then guess one $S_t^{-1}[z_t]$ for one of the equations belonging to the clique. Afterwards, the IR block is called recursively.

In Figure 3(f) the maximum clique is of size 4 equations with 5 unknowns. It means that a guess of only one unknown reveals four other ones. Furthermore, the space of possible guesses is singnificantly reduced due to the higher probability of a contradiction to occur.

Window Expansion (WE) Block Obviously, the more equations we have the faster the algorithm works. Therefore, a new equation is added to the system as soon as the missing value $S[i]$ in front or in back of the window is derived. The *Window Expansion* block checks for this event and dynamically extends the window. Sometimes several equations are added at once, especially on the leafs of the recursion.

Guess One $S[i]$ (GSi) Block If there are no active equations but the ground state S_0 is not yet fully determined, the window is then expanded by a direct guess of $S[i]$, in front or in back of the window. Then the WE, IR, MC blocks continue to work as usual. Additional heuristics can be applied for choosing which side of the window to be expanded for a larger success.

3 Precomputations: Finding Good Patterns

Assume at time t the internal state of RC4 is compliant to a certain pattern. An effectiveness of the new state recovering attack strongly depends on the properties of the pattern. If a pattern has a large window then it helps to decreases the complexity of the algorithm efficiently. However, it is less probable for the internal state to be compliant to a pattern with large number of conditions.

In this section we introduce various properties of patterns that influence on the attack success, and also study their availability.

3.1 Generative States

Let us start with the following definition

Definition 1 (d-order pattern). *A d-**order pattern** is a tuple*

$$A = \{i, j, P, V\}, \quad i, j \in \mathbb{Z}_N, \tag{5}$$

where P and V are two vectors from $\mathbb{Z}_N^d$ with pairwise distinct elements. At a time t the internal state is said to be ***compliant with*** *A if $i_t = i, j_t = j$, and d cells of the state S_t with indices from P contain corresponding values from V.* □

The example in Figure 3 illustrates how a 5-order pattern allows to receive a window of length 15. However, the higher the order, the less the probability of such a constraint to happen. Therefore, we are interested in finding a low order pattern which generates a long window.

Definition 2 (w-generative pattern). *A pattern A is called* ***w-generative*** *if for any internal state compliant with A the next w clockings allow us to derive w equations of the form (3), i.e., consecutive w values of js are known.* □

Table 1 demonstrates a 4-order 7-generative pattern $A = \{-7, -8, \{-6, -5, -4, 0\}, \{6, -1, 2, -2\}\}$, which supports the above definitions. Eight equations involve symbols of the keystream $z_{t+1}, \ldots, z_{t+8}$ associated with a certain time t. Here and further we say the ***keystream is true*** if the internal state at time t is compliant with the pattern, otherwise we say the ***keystream is random***.

Let another pattern B be derived from A as

$$B = A + \tau = \{i + \tau, j + \tau, P + \tau, V\}, \tag{6}$$

for some "shift" τ. The pattern B is likely to be w-generative as well. This happens when the properties of A are independent of N, which is the usual case.

i_t	j_t	$S[i]$	$S[j]$	$S[i]+S[j]$	z_t	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5
-7	-8	–	–	–	–	6	-1	2	x_1	x_2	x_3	-2	x_4	x_5	x_6	x_7	x_8
-6	-2	6	x_2	$6+x_2$	$*$	x_2	-1	2	x_1	6	x_3	-2	x_4	x_5	x_6	x_7	x_8
-5	-3	-1	x_1	$-1+x_1$	$*$	x_2	x_1	2	-1	6	x_3	-2	x_4	x_5	x_6	x_7	x_8
-4	-1	2	x_3	$2+x_3$	$*$	x_2	x_1	x_3	-1	6	2	-2	x_4	x_5	x_6	x_7	x_8
-3	-2	-1	6	5	x_8	x_2	x_1	x_3	6	-1	2	-2	x_4	x_5	x_6	x_7	x_8
-2	-3	-1	6	5	x_8	x_2	x_1	x_3	-1	6	2	-2	x_4	x_5	x_6	x_7	x_8
-1	-1	2	2	4	x_7	x_2	x_1	x_3	-1	6	2	-2	x_4	x_5	x_6	x_7	x_8
0	-3	-2	-1	-3	-2	x_2	x_1	x_3	-2	6	2	-1	x_4	x_5	x_6	x_7	x_8
1	$*$	x_4	$*$	$*$	$*$												

Table 1. An example of a 4-state 7-generative pattern.

3.2 Availability

We have done a set of simulations in order to find *maximum w-generative d-order patterns* (denoted by $\mathcal{M}_d$); see Table 7(a) for the results. Searching for a high order pattern is a challenging task since the computation complexity grows exponentially with d, and the best result achieved in our work is a 14-order 76-generative pattern $\mathcal{M}_{14}$.

	Real values from our simulations													Approximated values						
$d =$	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
$w_{\mathtt{max}} =$	6	10	15	21	27	31	37	42	50	55	61	68	76	82	88	94	100	106	112	118

Table 2. Dependency of the maximum w from d, simulated and approximated values.

Table 2 shows the dependency of a maximum achievable generativeness $w_{\mathtt{max}}$ from the order d. One may note that this dependency is almost linear, and it converges to $w_{\mathtt{max}} = 6d + \lambda$ as $d \to \infty$. Let us make the following conjecture.

Conjecture 1. We conjecture that the rate of $\frac{w_{\mathtt{max}}}{d} \approx 6$ as $d \to \infty$. $\square$

Indeed, the "jump" of $w_{\mathtt{max}}$ as d increments by one is the sequence $\{4,5,6,6,4,6,5,8,5,6,7,8,\ldots\}$. Obviously, for small ds this "jump" is small, and it is notable that the "jump" increases for larger ds. In our simulations we used heuristics (see Section 3.3) when searching patterns for $d \geq 6$, this means that that "jump" is possibly even larger at the sequence since our patterns found by heuristics are not optimal. This shows that the ratio $w \to 6d$ as $d \to \infty$ seems quite a fair conjecture.

This conjecture allows us to make a prediction about certain parameters for patterns with large ds, which we actually could not find due to a very high precomputation complexity, but they are needed for the attack for large Ns ($N = 128, 256$). Given those parameters (d and w) one can derive theoretical complexity of the attack in average, and it has been done in our work as well (see Appendix A).

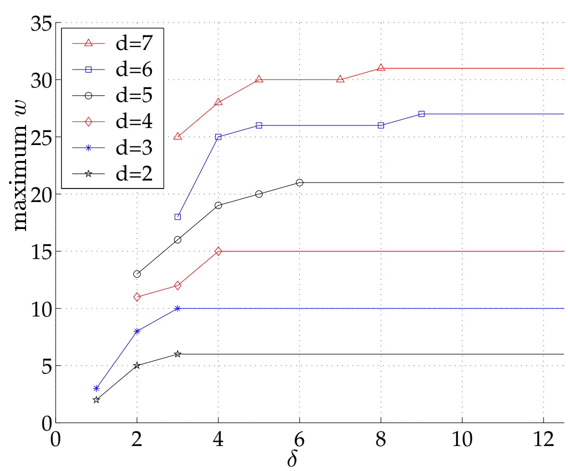

Fig. 4. Dependency of the maximum w from δ for various d.

As it can be seen from Table 7 all found “good” patterns have Vs with the values from a short interval $I_\delta = [-\delta \ldots +\delta]$, where $\delta \approx 10 \ldots 25$ is quite conservative. Figure 4 illustrates the dependency of the maximum achievable w from δ, and this allows us to make another conjecture.

Conjecture 2. A pattern with the largest w is likely found among all possible combinations for $i = 0, j \in I_\delta, V \in I_\delta^d$, with a moderate value of $\delta \ll N$. $\square$

This conjecture is the basis for a *significant* improvement in *searching technique* of such patterns (see Section 3.3). Table 3 provides the number of patterns for $\delta = 15$, and various values of d and w. When d and δ are fixed, the amount of desired patterns can exponentially be increased by letting w to be slightly less than $w_{\texttt{max}}$, and this can help to find patterns with additional properties which we will introduce in further sections.

d		The number of patterns A_d when $\delta = 15$.									
$\downarrow$	$w \rightarrow$	15	14	13	12	11	10	9	8	7	6
4	$\#\{A_4\} \rightarrow$	1	3	10	26	226	863	5234	21702	114563	853012
	$w \rightarrow$	21	20	19	18	17	16	15	14	13	12
5	$\#\{A_5\} \rightarrow$	1	4	6	15	66	252	652	1879	6832	27202
	$w \rightarrow$	27	26	25	24	23	22	21	20	19	18
6	$\#\{A_6\} \rightarrow$	1	2	7	42	81	177	371	799	2646	10159

Table 3. The number of different constrains for specific d and w, when $\delta = 15$.

3.3 Searching Technique

Since the searching space for a d-order pattern grows exponentially, only patterns of order $d \leq 6$ were analysed before in various literature. In this section we suggest a few techniques that accelerate the search significantly, and allow to search and analyse patterns of order up to $d \leq 15$, approximately, on a usual desktop PC.

The first idea is to set $i = 0$ due to (6), and for the remaining variables only a small set of values I_δ with some δ should be tested due to Conjecture 2.

A straightforward approach would be to allocate d values in a vector S and then to check the desired properties of the pattern. Its time complexity is $O\left(\binom{N}{d}\binom{|I_\delta|}{d}|I_\delta|\right)$, which is still very large. However, our *second idea* is to allocate a new element in S only when it is necessary.

The diagram of the recursive algorithm exploiting the first two ideas is shown in Figure 5, but it can be improved with the following *heuristic*. The *third idea* is to start searching for a desired pattern somewhere in the middle of its future window. Let us split d into $d_{\texttt{fwd}} + d_{\texttt{back}}$ and then start the algorithm in Figure 5 allowed to allocate exactly $d_{\texttt{fwd}}$ cells of S. At the point $(*)$ the current length of the window w is compared with some threshold $w_{\texttt{thr}}$. If $w \geq w_{\texttt{thr}}$, then a similar recursive algorithm starts, but it goes backward and allocates remaining $d_{\texttt{back}}$ cells of S. This double-recursion results in a pattern with w *likely* to be close to the maximum possible length of the window.

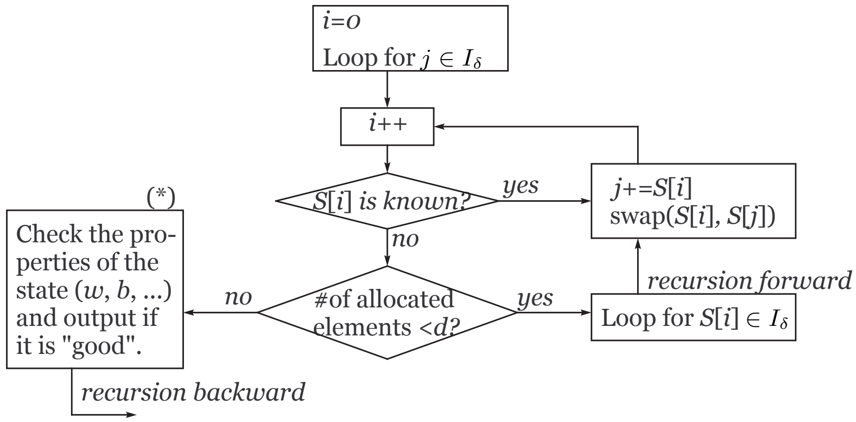

Fig. 5. Recursive algorithm for searching patterns with large w.

4 Detection of Patterns in the Keystream

In the previous section we have studied properties of a pattern that are desirable for the state recovering algorithm to work fast and efficient. We have also shown how these patterns can be found, and introduced an efficient searching algorithm.

In this section we show how the internal state of RC4, compliant to a chosen pattern, can be detected by observing the keystream. If that detection is very good, then the number of executions of the state recovering algorithm can be as small as just once, but at the right place of the keystream.

The detection mechanism itself can be trivial (no detection at all), in which case the algorithm has to be run at every position of the keystream. A good detection may require a deep analysis of the keystream, where specific properties of the pattern can be used efficiently.

4.1 First Level of Analysis

The internal state of RC4 compliant to a d-order pattern A can be regarded as an *internal event* with probability

$$\Pr\{E_{\mathtt{int}}\} = N^{-d-1}. \tag{7}$$

When the internal event happenes, there could exist an *external event* $E_{\mathtt{ext}}$ observed in the keystream, and associated with the pattern A, i.e., $\Pr\{E_{\mathtt{ext}}|E_{\mathtt{int}}\} = 1$. Applying the Bayes' law one can derive

$$\mathcal{P}_{\mathtt{det}} = \Pr\{E_{\mathtt{int}}|E_{\mathtt{ext}}\} = \frac{\Pr\{E_{\mathtt{int}}\}}{\Pr\{E_{\mathtt{ext}}\}}, \tag{8}$$

and this is precisely the *detection probability* of the pattern A in the keystream. Our goal in this section is to study possible external events with high $\mathcal{P}_{\mathtt{det}}$ in order to increase the detection of the pattern.

Definition 3 (l-definitive pattern). *A w-generative pattern A is called l-**definitive** if there are exactly l out of w equations with determined $S[j]$s.* $\square$

It means that in l equations $S[i]+S[j]$ is known. If, additionally, $z' = S[S[i]+S[j]]$ is also known, then the correct value of $z_t = z'$ at the right place t of the keystream $\mathbf{z}$ detects the case "*the state at time t* **is possibly** *compliant to the pattern*". Otherwise, when $z_t \neq z'$, it says that "*the state at time t* **cannot be** *compliant to the pattern*".

For detection purposes a large l (up to d) is important. From our experiments we found that, however, it can be achieved via a slight reduction of the parameter w, and it leads us to one more conjecture.

Conjecture 3. For any d and $w = w_{\mathtt{max}} - \lambda$ there exist a pattern with $l = d$, where λ is relatively small. $\square$

Table 6(a) contains patterns $\mathcal{X}$s with $l = d$ where w is still large, which supports the above conjecture.

Definition 4 ($b_\alpha, b_\beta, b_\gamma$-$^{\alpha,\beta,\gamma}$predictive pattern). *Let us have an l-definitive pattern A and we consider only those equations where $S[j]$s are determined. Then, the pattern A is called b_α-$^{\alpha}$**predictive** if for b_α of the l equations $S[S[i]+S[j]]$ is determined. For the remaining $l-b_\alpha$ equations two additional definitions are as follows. The pattern A is called b_β-$^{\beta}$**predictive** if for b_β pairs of the $l-b_\alpha$ equations the unknowns $S[S[i]+S[j]]$s must be the same. The set of b_β pairs must be of full rank. The pattern A is called b_γ-$^{\gamma}$**predictive** if the $l-b_\alpha$ equations contain exactly b_γ different variables of $S[S[i]+S[j]]$.* $\square$

These types of predictiveness are other properties of a pattern visible in the keystream. For example, it is not only necessary to search for known z' values (b_α of such), but one can also require that certain pairs of the keystream symbols (b_β of such) are equal $z_{t'} = z_{t''}$, which also helps to detect the pattern significantly.

The parameter b_α is usually quite moderate, to have it larger than 15 is quite difficult. However, the other criteria are more flexible and can be large. These new parameters follow the constraint

$$b_\alpha + b_\beta + b_\gamma = l \leq d. \tag{9}$$

Consider the remaining $w-l$ equations of the pattern A where $S[j]$s are not determined. Let at times t_1 and t_2 one pair of these equations be such that the values $S[i]$s and the pointers $S[j]$s are equal. If the distance $\Delta_t = t_2 - t_1$ is small, it is likely that the output z_1 is the same as z_2. The probability of this event is

$$\Pr\{z_1 = z_2 | \Delta_t\} > \left(1 - \frac{\Delta_t}{N}\right) \cdot \left(1 - \frac{1}{N}\right)^{\Delta_t} \approx \exp\left(-\frac{2\Delta_t}{N}\right). \tag{10}$$

Definition 5 **(b_θ-$^\theta$predictive pattern).** *A pattern A is called b_θ-$^\theta$**predictive** if the number of such pairs (described above) is b_θ. Let the time distances of these pairs be $\Delta_1, \ldots, \Delta_{b_\theta}$, then the **cumulative distance** is the sum $\Pi_\theta = \Sigma_i \Delta_i$* □

These four types of predictiveness are direct external events for a pattern. One should observe the keystream and search for certain b_α symbols, check another b_β and b_θ pairs of symbols that they are equal, and also check that a group of b_γ symbols are different from the values of V and from each other. Thus, we have

$$\begin{aligned} \Pr\{E_{\mathtt{ext}}\} &= N^{-b_\alpha - b_\beta - b_\theta} \cdot \left[\frac{(N-d)!}{N^{b_\gamma}(N-d-b_\gamma)!}\right] \\ \Pr\{E_{\mathtt{int}}\} &\approx N^{-d-1} \cdot e^{-2\Pi_\theta/N}. \end{aligned} \tag{11}$$

The example in Table 1 is a 4-definitive $b_\alpha = 1, b_\beta = 1, b_\gamma = 2, b_\theta = 0$-predictive pattern. For detection one has to test that $z_{t+6} = -2, z_{t+3} = z_{t+4}$, and z_{t+4}, z_{t+5} are different from the initial values at V and $z_{t+4} \neq z_{t+5}$. I.e., when, for example, $N = 64$, the detection probability is $64^{-5} \div (64^{-2} \cdot 60 \cdot 59/64^2) \approx 64^{-2.96}$ [2].

4.2 Second Level of Analysis

In fact, the first level of analysis allows to detect a pattern with probability at most N^{-1} (because of j is not detectable), whereas with the second level of analysis it can be 1. Let us introduce the technique that we called a *chain of patterns.*

Definition 6 **(chain of patterns $A \to B$, distance, intersection).** *Let us have two patterns $A = \{i_a, j_a, P_a, V_a\}$ and $B = \{i_b, j_b, P_b, V_b\}$. An event when two patterns appear in the keystream within the shortest possible time distance σ is called **chain of patterns**, and denoted as $A \to B$ if B appears after A.*

*The **chain distance** σ between two patterns A and B is the shortest possible time between A's ending and B' beginning of their windows, i.e.,*

$$\sigma = i_b - (i_a + w_a) \mod N. \tag{12}$$

*The **intersection** of A and B is the number ξ of positions in A that are reused in B. These positions must not appear as $S[i]$ during σ clockings while the chain distance between A and B is approached.* □

For example, let $A = \{0, 0, \{1, 3, 5, 6, 7, 8, 22, 23\}, \{2, 8, -3, -2, 1, 7, 4, -9\}\}$ and $B = \{34, 34, \{35, 36, 37, 38, 39, 44, 48, 52\}, \{8, -2, 1, 2, 4, -5, 5, 3\}\}$. After $w_a = 30$ clockings the first pattern becomes $A' = \{30, 28, \{15, 28, 30, 35, 36, 37, 38, 39\}, \{-3, -9, 7, 8, -2, 1, 2, 4\}\}$. Obviously, the last $\xi = 5$ positions can be reused in

[2] Since $^\gamma$-predictiveness has a minor influence on detection, further we skip this parameter in calculations.

B, and after $\sigma = 4$ clockings a new pattern B ($w_b = 34$) can appear if $j_{t+34} = j_b$. The probability of the chain $A \to B$ to appear is $N^{-9} \cdot N^{-4}$, times the probability that 5 elements from A' are in place during 4 clockings, and it is much larger than the trivial $N^{-9} \cdot N^{-9}$. Thus, a more general theorem can be stated.

Theorem 1 (chain probability). *The probability of a chain $A \to B$ to appear is*

$$\mathcal{P}_{A\to B} = \Pr\{E_{\mathtt{int}}\} \approx N^{-(d_a+d_b+2-\xi)} \cdot e^{-2(\Pi_{\theta a}+\Pi_{\theta b})/N} \cdot e^{-\xi}. \tag{13}$$

Proof. In [Man01] it has been shown that ξ elements stay in place during N clockings with an approximate probability $e^{-\xi}$. The remaining part comes from an assumption that the internal state is random, from where the proof follows. □

Obviously, the probability of the external event for the chain is

$$\Pr\{E_{\mathtt{ext}}\} = N^{-(b_{\alpha a}+b_{\beta a}+b_{\theta a})-(b_{\alpha b}+b_{\beta b}+b_{\theta b})}, \tag{14}$$

which can be smaller than $\Pr\{E_{\mathtt{int}}\}$ (see $\mathfrak{Y}_4$ in Table 6), confusing the equation (8). This happens since $\Pr\{E_{\mathtt{ext}}\}$ is calculated assuming that the keystream is random. However, in real RC4 only a portion of the observed external probability space can appear (which is another source for a distinguishing attack, but it is out of scope of this paper). Therefore, in case when $\Pr\{E_{\mathtt{ext}}\} < \Pr\{E_{\mathtt{int}}\}$ we simply assume that the detection probability is 1.

Table 6 presents only several examples with a good trade-off (based on our intuition) between w and detectability for various d. Since the computation time for searching such patterns with multiple desired properties is really huge, only a few examples for small ds were given. However, we believe that for large ds it is possible to detect such patterns with a high probability, up to 1, applying two proposed levels of analysis.

5 Complete State Recovering Attack on RC4

5.1 Attack Scenario and Total Complexity

Recall pattern detection techniques from Section 4. In the complete attack scenario an adversary analyses the keystream at every time t, and applies the state recovering algorithm if the desired internal event (pattern) is detected. In all cases except one the recovering algorithm deals with a random keystream.

Proposition 1 (Total Attack Complexities). *Let the detection probability be $\mathcal{P}_{\mathtt{det}}$, then the total time C_T and data C_D complexities of the attack are*

$$\begin{aligned} C_T &= \Pr\{E_{\mathtt{int}}\}^{-1} + (\mathcal{P}_{\mathtt{det}}^{-1} - 1) \cdot C_{\mathtt{Rand}} + 1 \cdot C_{\mathtt{True}}, \\ C_D &= \Pr\{E_{\mathtt{int}}\}^{-1}. \end{aligned} \tag{15}$$

□

5.2 Success Rate of the Attack

The complexities $C_{\mathtt{True}}$ and $C_{\mathtt{Random}}$ are upper bounds for the *average* time the algorithm works. It means that for some cases it could take more time than these bounds. In order to guarantee the upper bound of the total (not average) time complexity one can terminate the algorithm after, for example, $C_{\mathtt{thr}}$ operations are done. In this case the *success rate* of the attack can be determined.

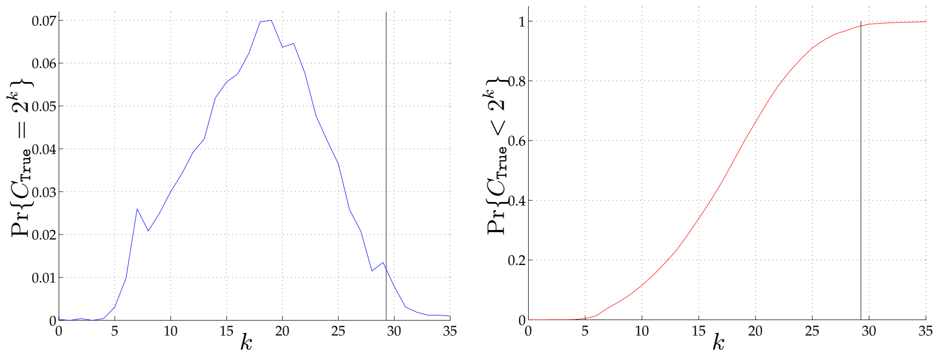

Fig. 6. Probability density (left) and cumulative (right) functions of the time $C_{\mathtt{True}}$ in the logarithmical form ($k = \log_2 C_{\mathtt{True}}$). The scenario is $N = 64, \mathcal{M}_8$, 2000 samples.

Figure 6 shows density and cumulative functions for the time complexity of an example attack scenario. It shows that around 98% of all simulations of the attack have time smaller than the average $2^{29.28}$ (vertical line). When the keystream is random the termination makes the average time bound $C_{\mathtt{Random}}$ even smaller, since the random case is likely to be repeated very many times and the second term in (15) can only decrease.

The plots in Figure 6 also show that even if the termination of the algorithm is done on the level $C_{\mathtt{thr}} = \sqrt{C_{\mathtt{True}}}$ ($\approx 2^{15}$), the success rate of the algorithm is still very high. I.e., the state recovering algorithm on RC4-64 can be done in time 2^{15} with success probability 35%! If a similar situation happens for large Ns ($N = 256$, etc), then the full time complexity can be significantly decreased (perhaps, down to a square root of the estimated average complexity), and the success probability can still be very large.

6 Simulation Results and Conclusions

We have selected a set of test cases with various parameters and patters, and derived total data and time complexities of the new attack. Table 4 presents the results of this work. For example, when $N = 64$, the total complexity of the new attack is upper bounded by 2^{60}, if the pattern $\mathcal{X}_9$ is used. This is much faster than, for example, Knudsen's attack whose complexity for this case is $2^{132.6}$. Even if $d = 9$ elements of the state are known, Knudsen's attack needs $2^{98.1}$ of time, which is still much higher. The complexity of a potential (not really working [3]) attack recently discussed by I. Mantin in [Man05] is also higher. As it was shown in Section 5.2, the success rate of the new attack is at least 98%, and further we simply assume it is close to 100%.

	N	$N=64$			$N=100$		$N=128$		$N=160$		$N=200$		$N=256$	
	Cases	I	II	III	IV	V	VI	VII	VIII	IX	X	XI	XII	XIII
		Descriptions of the cases (⋆ – are hypothetical cases)												
	Pattern	$\mathcal{M}_8$	$\mathcal{Y}_8$	$\mathcal{X}_9$	$\mathcal{X}_{11}$	$\mathcal{M}_{13}$	$\mathcal{M}_{14}$	⋆	$\mathcal{M}_{14}$	⋆	$\mathcal{M}_{14}$	⋆	$\mathcal{M}_{14}$	⋆
	d	8	8	9	11	13	14	17	14	18	14	23	14	29
	w	37	29	41	49	68	76	92	76	102	76	132	76	168
	l	6	6	5	11	9	10	10	10	10	10	14	10	17
	b_α	0	4	4	9	0	0	10	0	11	0	10	0	11
	b_β	1	1	0	0	2	2	0	2	0	2	2	2	4
	b_γ	5	1	1	2	7	8	0	8	0	8	2	8	2
	b_θ	0	0	2	0	2	2	0	2	7	2	4	2	12
	Π_θ	0	0	4	0	4	4	0	4	–	4	–	4	–
		Internal/external/detection probabilities												
	$\mathcal{P}_{\mathtt{int}}$	-54.0	-65.8	-60.0	-79.7	-93.0	-105.0	-112.0	-109.8	-139.1	-114.7	-183.5	-120.0	-240.0
	$\mathcal{P}_{\mathtt{ext}}$	-6.0	-60.0	-36.0	-59.8	-26.6	-28.0	-70.0	-29.3	-131.8	-30.6	-122.3	-32.0	-216.0
	$\mathcal{P}_{\mathtt{det}}$	-48.0	-5.8	-24.0	-19.9	-66.4	-77.0	-42.0	-80.5	-7.3	-84.1	-61.2	-88.0	-24.0
		Complexities of the state recovering algorithm when the keystream is true/random												
$C_{\mathtt{Rand}}$	Theor.	20.5	58.2	22.8	107.8	10.0	71.3	71.7	191.1	131.7	317.4	121.3	507.4	217.1
	Attun.	15.5	57.8	–	107.5	–	66.3	–	179.2	–	302.6	–	491.8	–
$C_{\mathtt{True}}$	Theor.	35.0	64.9	30.9	120.4	34.5	94.7	102.0	213.0	138.2	335.6	157.5	519.6	225.4
	Attun.	30.3	57.6	–	108.3	31.8	85.5	–	185.1	–	309.9	–	501.8	–
	Real	*29.3*	–	–	–	*29.1*	–	–	–	–	–	–	–	–
		Total data/time complexity, and the comparison with previous attacks												
Knudsen's	$C_{\mathtt{K}}(0)$	**132.6**			**236.6**		**324.8**		**431.4**		**572.0**		**779.7**	
	$C_{\mathtt{K}}(d)$	101.7	101.7	98.1	189.3	181.0	261.3	256.9	364.6	346.1	501.9	458.2	705.9	629.3
Mantin's *potential attack*		**73**			**114**		**147**		**186**		**243**		**290**	
Our attack	C_D	54.0	54.0	**60.0**	79.8	**93.0**	105.0	**112.0**	109.8	**139.1**	114.7	**183.4**	120.0	**240.0**
	C_T	63.5	63.4	**60.0**	127.4	**93.1**	143.4	**113.7**	271.7	**140.4**	386.7	**184.0**	579.8	**241.7**

Table 4. Simulation results and comparisons with previous attacks.

Table 4 also contains intermediate probabilities and complexities of the attack, including theoretical ($\Delta = 0$) and attuned ($\Delta = 2$) values for $C_{\mathtt{Rand}}$ and $C_{\mathtt{True}}$. When it was possible, the real attack *on a true keystream* was simulated (real complexities for $C_{\mathtt{True}}$ are shown in italic). In these simulations the complete

[3] In Mantin's attack they detect a large number of bytes of the state, and then apply Knudsen's attack given those bytes. However, to make these knowns to reduce the attack complexity they must be located in a short window all together, and this is not the case.

state of RC4 was successfully recovered for every randomly generated keystream compliant with the corresponding pattern.

For larger Ns patterns of a high order are needed to receive an attack of a low complexity. The largest pattern that we could find in this work is $\mathcal{M}_{14}$, and it was applied to attack RC4-N with $N = 128, 160, 200, 256$. These attack scenarios are those that we have in our hands already. However, the complexities received are not optimal, but they are still lower than in Knudsen's attack. Conjecture 1 and also discussions in Section 4 make it possible to approximate the parameters of a hypothetical pattern that is likely to exist ($\star$ – patterns). To be secure, we relate d and w as $w = 6d - 6$, with a confidence gap of 6 positions. The remaining parameters were chosen moderate as well. As the result, we obtained an attack on RC4-256 with the (upper bounded) total complexity of $2^{241.7}$, and this is the best state recovering attack known to the moment.

In general, we have noted the following tendency. For RC4-N with a secret key of length N bits or longer, the new attack can recover the internal state much faster than an exhaustive search. This observation can also be seen from the results in Table 4.

As the last point of the discussions we should mention the existance of various papers which deal with a recovering of the secret key conditioned that the internal state of RC4 is known (initialisation of RC4 is not one-way). This part is significantly faster than any of the the state recovering algorithm, and, therefore, we just refer to these papers [MS01,Man01,PM07].

7 Further Improvements and Open Problems

Pattern detection improvements. With a chain of patterns described in Section 4 one could reach a good detection. However, not only forward direction of chaining can be considered, but also backward one. Additionally, there is a possibility to analyse longer sequences of patterns in order to have a good detectability.

Another idea is to use *unusual recyclable patterns* in a similar manner as in [Man05]. The difference is that these patterns are both recyclable and have a long window. For example, $A = \{0, -4, \{6, 4, 1, 5, 3\}, \{0, 1, 7, -2, -1\}\}$.

State recovering algorithm improvement. The GSi block can choose the corner (left or right) of the window to be extended by an additional heuristic analysis of the current situation during the process. Another improvement is achieved if the MC block could speculatively run the recursion for additional 1-3 extra forward steps for every possible guess, and, afterwards, make such a guess for which the number of sub branches is the minimum. The average time of the attack for this strategy is reduced.

Derivation and statistics. Our investigation showed that the derived theoretical upper bound gives a much larger complexity than the one received from the real simulations of the attack. Obviously, a better analysis of the algorithm's complexity is needed, this would allow to estimate total complexities more accurate, and it might improve the complexities in Table 4 significantly. Another interesting problem is to determine the density function of the recovering algorithm, likewise in Figure 6. This may allow us to decrease the complexity in square root times, and still the success rate will be very high.

Other open problems. The search for patterns of a higher order with long windows is another challeging open question. We have shown that there are chains of patterns with short distances. The first pattern is used for the recovering algorithm, and the second one is for detection. However, here is another interesting question whether the second pattern can also be used in the recovering algorithm or not.

We believe that the outlined open problems have a huge potential for reducing the complexity of the attack on RC4. Perhaps, very soon we will be witnessing an attack of complexity lower than 2^{128} *on the full RC4-256.*

References

[FM00] S. R. Fluhrer and D. A. McGrew. Statistical analysis of the alleged RC4 keystream generator. In B. Schneier, editor, *Fast Software Encryption 2000*, volume 1978 of *Lecture Notes in Computer Science*, pages 19–30. Springer-Verlag, 2000.

[Gol97] J. Dj. Golić. Linear statistical weakness of alleged RC4 keystream generator. In W. Fumy, editor, *Advances in Cryptology—EUROCRYPT'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 226–238. Springer-Verlag, 1997.

[KMP+98] L. R. Knudsen, W. Meier, B. Preneel, V. Rijmen, and S. Verdoolaege. Analysis methods for (alleged) RC4. In K. Ohta and D. Pei, editors, *Advances in Cryptology—ASIACRYPT'98*, volume 1998 of *Lecture Notes in Computer Science*, pages 327–341. Springer-Verlag, 1998.

[Man01] I. Mantin. Analysis of the stream cipher RC4. Master's thesis, The Weizmann Institute of Science, Department of Applied Math and Computer Science, Rehovot 76100, Israel., 2001.

[Man05] I. Mantin. Predicting and distinguishing attacks on RC4 keystream generator. In R. Cramer, editor, *Advances in Cryptology—EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 491–506, 2005.

[Max05] A. Maximov. Two linear distinguishing attacks on VMPC and RC4A and weakness of RC4 family of stream ciphers. In H. Gilbert and H. Handschuh, editors, *Fast Software Encryption 2005*, volume 3557 of *Lecture Notes in Computer Science*, pages 342–358. Springer-Verlag, 2005.

[MS01] I. Mantin and A. Shamir. Practical attack on broadcast RC4. In M. Matsui, editor, *Fast Software Encryption 2001*, volume 2355 of *Lecture Notes in Computer Science*, pages 152–164. Springer-Verlag, 2001.

[MT98] S. Mister and S. E. Tavares. Cryptanalysis of RC4-like ciphers. In *Selected Areas in Cryptography—SAC 1998*, Lecture Notes in Computer Science, pages 131–143, 1998.

[PM07] G. Paul and S. Maitra. Rc4 state information at any stage reveals the secret key. Available at *http://eprint.iacr.org/2007/208* (accessed January 10, 2008), 2007.

[PP04] S. Paul and B. Preneel. A new weakness in the RC4 keystream generator and an approach to improve the security of the cipher. In B. Roy and

W. Meier, editors, *Fast Software Encryption 2004*, volume 3017 of *Lecture Notes in Computer Science*, pages 245–259. Springer-Verlag, 2004.

[Sma03] N. Smart. *Cryptography: An Introduction*. McGraw-Hill Education, 2003. ISBN 0-077-09987-7.

A Complexity Analysis of the Recovering Attack

Since for large inputs it is not always possible to make real simulations of the new recovering attack, one is interested in a theoretical upper bound of its complexity. In this section we explain how this complexity can be derived, verified and used.

A.1 Tool for Simulations and Analysis

The new recovering algorithm is a recursion as shown in Figure 7(a). The nodes are IR and WE blocks, whereas each branch is initiated by MC or GSi blocks. A branch is terminated when a contradiction occurs, and only one path leads to the correct solution, where the internal state is successfully recovered.

We measure the complexity of the attack as the number of branches, i.e., the number of guesses in the MC and GSi blocks done.

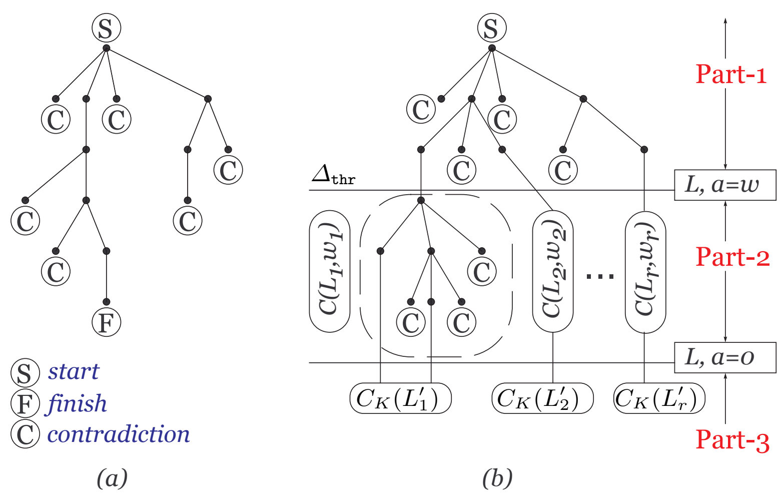

Fig. 7. (a) Attack as a recursion; (b) Three parts of the tool for simulations.

Let us introduce a three-parts tool, shown in Figure 7(b), in order to calculate the complexity of the attack when a certain pattern is given. The description of these parts is the following.

In the *first part* the simulation of the attack with a certain pattern is launched (all four blocks, IR, WE, MC, GSi, are working), and the number of branches is counted. Whenever the depth of the recursion becomes $\Delta_{\mathtt{thr}}$, some precomputed function for the complexity of the remaining subtree is called, and the recursion makes a backward step.

The *second part* is a precomputed pattern-independent upper bound of the average complexity, when the status of the recursion can be described as the number of already allocated cells L and the number of active equations a.

The *third part* is Knudsen's attack complexity accepted as an upper bound for the algorithm on the leafs of the recursion, in order to avoid analysis of WE.

To receive theoretical complexity using this tool one should run the simulations for a sufficient number of times, and then take an average of the results. The exact complexity is received when $\Delta_{\mathtt{thr}} = \infty$, in this case the tool works the same time as the targeting complexity. On the other hand, when $\Delta_{\mathtt{thr}} = 0$, the upper bound of the complexity is received immediately. The reason to introduce $\Delta_{\mathtt{thr}}$ and the three parts of the tool will be explained later.

A.2 Assumptions

We will derive the precomputed pattern-independent upper bound of the average complexity under the following assumptions.

Assume that the algorithm first processes all given w equations of the kind (3) with two unknowns in each, and then Knudsen's attack is applied to the remaining part of the recursion (see in the right table on the columns with WE on and off).

Assume that in all given w equations the values $S_t[j_t]$ refer to *different* unknowns. This makes the attack to work slower since in the MC block the maximum clique can, therefore, be constructed only via symbols of the keystream. Table on the right shows that for this assumption the complexity of the attack is larger.

Assume that the keystream is random, which is reasonable since the real internal state is unknown to an attacker.

We have selected several patterns with similar properties, $d = 4, w = 9$ ($\mathfrak{A}$s and $\mathfrak{B}$s from Table 7). One half of them have different $S_t[j_t]$s, and the other half contains pairs of equal $S_t[j_t]$s. Afterwards, the complexities of the

(Logarithms of the complexities)				
	Random $\mathbf{z}$		True $\mathbf{z}$	
Patrn.	WE off	WE on	WE off	WE on
$\Delta_{\mathtt{thr}} = \infty, N = 25, d = 4, w = 9,$ *all* $S_t[j_t]$ *are different.* # of tests is ≥ 500.				
$\mathfrak{A}_1$	15.87	14.25	16.33	15.09
$\mathfrak{A}_2$	15.24	14.02	16.30	14.38
$\mathfrak{A}_3$	14.89	14.48	16.00	14.80
$\mathfrak{A}_4$	15.51	14.18	16.38	14.44
$\mathfrak{A}_5$	15.20	12.97	15.87	12.57
$\mathfrak{A}_6$	14.98	12.02	15.50	11.66
Average	15.32	13.86	16.09	14.24
$\Delta_{\mathtt{thr}} = \infty, N = 25, d = 4, w = 9,$ *at least two* $S_t[j_t]$ *coincide.* # of tests is ≥ 5000.				
$\mathfrak{B}_1$	7.41	7.95	13.08	13.49
$\mathfrak{B}_2$	5.08	3.71	13.42	12.03
$\mathfrak{B}_3$	4.62	3.67	13.30	12.00
$\mathfrak{B}_4$	4.84	4.43	10.28	10.06
$\mathfrak{B}_5$	3.41	3.72	11.42	12.21
$\mathfrak{B}_6$	2.94	3.19	12.00	13.38
$\mathfrak{B}_7$	3.81	4.57	11.12	12.39
Average	5.37	5.60	12.48	12.54

Assumptions make the algorithm slower and bound the real complexity.

attack are estimated ($\Delta_{\mathtt{thr}} = \infty, N = 25$) when the keystream is random/true, and WE is on/off. The results clearly show that the complexities under our assumptions are upper bounds.

A.3 Average Complexity Derivations

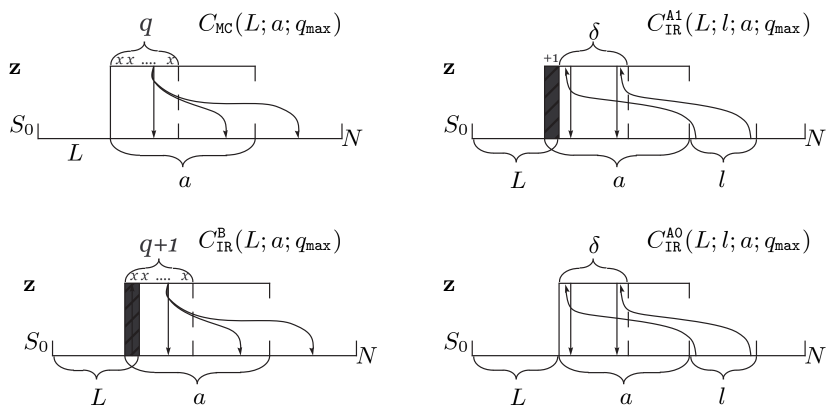

Fig. 8. Schematical four cases supporting derivations of the attack complexity.

In this section *a precomputed pattern-independent upper bound of the average complexity* is derived under the assumptions as proposed above. In all formulas the following meaning of variables is accepted: a is the number of active (not yet processed) equations of the form (3); L is the number of known and previously assigned cells of the state, and no single z_t from the active equations can be one of the L values; l is the number of recently (just) assigned cells of the state, and z_ts from active equations could possibly be one of the l values; $q_{\mathtt{max}}$ is the size of the maximum possible clique that can be found in the MC block.

Every step of the recursion has a complexity to which we will refer as: $C_{\mathtt{K}}(L)$ – is the complexity of Knudsen's attack, given that L cells of the internal state are known, and it can be precomputed as in [KMP+98]; $C_{\mathtt{MC}}(L;a;q_{\mathtt{max}})$ – is the complexity of the MC block; $C_{\mathtt{IR}}^{\mathtt{A0}}(L;l;a;q_{\mathtt{max}})$ – is the complexity of *one iteration* of the IR block that starts with L known and l new values, and ends with another set of new values of some size δ; $C_{\mathtt{IR}}^{\mathtt{A1}}(L;l;a;q_{\mathtt{max}})$ – the same as $C_{\mathtt{IR}}^{\mathtt{A0}}$, but for one of the equations the value of $S[j]$ is known; $C_{\mathtt{IR}}^{\mathtt{B}}(L;a;q_{\mathtt{max}})$ – the complexity of the case when IR returns none of new assignments, but for one equation $S[j]$ is known, i.e., the IR block makes an iteration of a different sort in this case.

Supplementary Formulas When L cells of S_0 are already known and δ new assignments are performed one-by-one, the probability of *no contradiction* will appear is

$$\mathcal{P}_c(L;\delta) = \frac{(N-L)!}{(N-L-\delta)!N^{\delta}}, \quad \text{when} \ \ 0 \le L+\delta \le N. \tag{16}$$

Let $M(r;a;q)$ be the number of possible keystream sequences of length a, where each symbol can have one out of r values, and the maximum possible size of a clique is q. The value of M can recursively be calculated as [4]

$$\begin{aligned} &M(r;a;q) = \sum_{i=0}^{q} \binom{a}{i} M(r-1;a-i;q), \quad \text{where} \ \begin{cases} 1 \le a,t \le N, \\ q \le a, \end{cases} \\ &M(r;0;0) = 1, \quad \text{where} \ \ 1 \le t \le N. \end{aligned} \tag{17}$$

Complexity $C_{\mathtt{IR}}^{\mathtt{A0}}(L;l;a;q_{\mathtt{max}})$ The probability that in *one iteration* δ out of a equations will be solved is

$$\begin{aligned} \mathcal{P}_{\mathtt{A0}}(L;l;a;\delta;q_{\mathtt{max}}) = &\binom{a}{\delta} \frac{M(l;\delta;q_{\mathtt{max}}) \cdot M(N-L-l;a-\delta;q_{\mathtt{max}})}{M(N-L;a;q_{\mathtt{max}})}, \\ &\text{when} \ \begin{cases} 0 \le L+l+a \le N, \\ 0 \le \delta \le a. \end{cases} \end{aligned} \tag{18}$$

In these δ equations z_t must be one of the l values and they must give δ *new* values $S_t[j_t]$, since, otherwise, they would have been found before. For each of the δ equations $S_t[z_t]$ is allocated somewhere, therefore, a *new* value $S_t[j_t] = S_t^{-1}[z_t] - S_t[i_t]$ can be derived. The number of active equations is evidently

[4] One should start with a loop for $t = 1 \to N$, and then a loop for $a = 1 \to N$, and then calculate the corresponding subtable.

reduced by δ. The total complexity of $C_{\mathtt{IR}}^{\mathtt{AO}}$ is recursively expressed as

$$\begin{aligned}
C_{\mathtt{IR}}^{\mathtt{AO}}(L;l;a;q_{\mathtt{max}}) = &\sum_{\delta=1}^{a-1} \mathcal{P}_{\mathtt{AO}}(L;l;a;\delta;q_{\mathtt{max}}) \cdot \mathcal{P}_c(L+l;\delta) \cdot C_{\mathtt{IR}}^{\mathtt{AO}}(L+l;\delta;a-\delta;q_{\mathtt{max}}) \\
&+ \mathcal{P}_{\mathtt{AO}}(L;l;a;a;q_{\mathtt{max}}) \cdot \mathcal{P}_c(L+l;a) \cdot C_{\mathtt{K}}(L+l+a) \\
&+ \mathcal{P}_{\mathtt{AO}}(L;l;a;0;q_{\mathtt{max}}) \cdot C_{\mathtt{MC}}(L+l;a), \quad \text{when } \begin{cases} 0 \le L+l+a \le N, \\ 1 \le q_{\mathtt{max}} \le a, \end{cases} \\
C_{\mathtt{IR}}(L;l;0;0) = &\ C_{\mathtt{K}}(L+l), \quad \text{when } \ L+l \le N.
\end{aligned} \tag{19}$$

Complexity $C_{\mathtt{MC}}(L;a;q_{\mathtt{max}})$ The probability of a maximum clique of size q to appear is

$$\mathcal{P}_{\mathtt{MC}}(L;a;q_{\mathtt{max}};q) = \frac{M(N-L;a;q) - M(N-L;a;q-1)}{M(N-L;a;q_{\mathtt{max}})}, \quad \text{where } \begin{cases} 1 \le L+a \le N, \\ 1 \le q \le q_{\mathtt{max}} \le a, \end{cases} \tag{20}$$

with a boundary case $\mathcal{P}_{\mathtt{MC}}(L;0;0;0) = 1$. The parameter $q_{\mathtt{max}}$ tells that in the remaining active equations no cliques of size more than $q_{\mathtt{max}}$ exist, since, otherwise, it would have been found on a previous call of the MC block.

Consider the unknown $x = S_t^{-1}[z_t]$ from the clique that has to be guessed as one of the $N-L$ remaining values. The choice of x is in principal one of the following three options. (a) x is one of the j_ts and the equation associated with time t belongs to the clique. It happens in q choices and results in $q-1$ new values. An additional contradiction test should be included: $S_t[i_t] + z_t$ must be equal to $S_t^{-1}[z_t]$ $(= x)$. (b) x is one of the j_ts and the equation associated with time t does not belong to the clique. It happens in $a-q$ choices and results in $q+1$ new values. (c) In the remaining $N-L-a$ choices q new values of the state are obtained.

Finally, the MC block is the only the block where the complexity is summarized. Thus, its total complexity is

$$\begin{aligned}
C_{\mathtt{MC}}(L;a;q_{\mathtt{max}}) = &\underbrace{(N-L)}_{\text{complexity}} + \sum_{q=1}^{q_{\max}} \mathcal{P}_{\mathtt{MC}}(L;a;q_{\mathtt{max}};q) \cdot \Big[\\
&+ \underbrace{q}_{q \text{ branches}} \cdot \underbrace{\frac{1}{N}}_{z_t = j_{t-1}} \cdot \mathcal{P}_c(L+1;q-1) \cdot C_{\mathtt{IR}}^{\mathtt{AO}}(L+1;q-1;a-q;q) \\
&+ \underbrace{(a-q)}_{a-q \text{ branches}} \cdot \mathcal{P}_c(L+1;q) \cdot C_{\mathtt{IR}}^{\mathtt{A1}}(L+1;q;a-q;q) \\
&+ \underbrace{(N-L-a)}_{\text{remaining branches}} \cdot \mathcal{P}_c(L+1;q) \cdot C_{\mathtt{IR}}^{\mathtt{AO}}(L+1;q;a-q;q) \Big], \\
&\text{when } \ 1 \le L+a \le N, \ \text{ and } \ 1 \le q_{\mathtt{max}} \le a.
\end{aligned} \tag{21}$$

Complexity $C^{\mathtt{A1}}_{\mathtt{IR}}(L;l;a;q_{\mathtt{max}})$ This case is similar to that of $C^{\mathtt{A0}}_{\mathtt{IR}}$, although there are two subcases with respect to the number of processed equations.

$$\begin{aligned}
C^{\mathtt{A1}}_{\mathtt{IR}}(L;l;a;q_{\mathtt{max}}) = \sum_{\delta=0}^{a-1} & \underbrace{\binom{a-1}{\delta} \frac{M(l;\delta;q_{\mathtt{max}})\cdot M(N-L-l;a-\delta;q_{\mathtt{max}})}{M(N-L;a;q_{\mathtt{max}})\cdot}}_{\text{probability of processing } \delta \text{ equations, except ``special'' one}} \\
& \times \mathcal{P}_c(L+l;\delta)\cdot \begin{Bmatrix} C^{\mathtt{B}}_{\mathtt{IR}}(L+l+\delta;a-\delta;q_{\mathtt{max}}), & \delta = 0, a-1 \\ C^{\mathtt{A1}}_{\mathtt{IR}}(L+l;\delta;a-\delta;q_{\mathtt{max}}), & \text{otherwise} \end{Bmatrix} \\
& + \sum_{\delta=0}^{a-1} \underbrace{\binom{a-1}{\delta} \frac{M(l;\delta+1;q_{\mathtt{max}})\cdot M(N-L-l;a-\delta-1;q_{\mathtt{max}})}{M(N-L;a;q_{\mathtt{max}})\cdot}}_{\text{probability of processing } \delta+1 \text{ equations, including ``special'' one}} \\
& \times \frac{1}{N}\cdot \mathcal{P}_c(L+l;\delta)\cdot \begin{Bmatrix} C_{\mathtt{MC}}(L+l;a-1;q_{\mathtt{max}}), & \delta=0 \\ C_{\mathtt{K}}(L+l+a-1), & \delta = a-1 \\ C^{\mathtt{A0}}_{\mathtt{IR}}(L+l;\delta;a-\delta-1;q_{\mathtt{max}}), & \text{otherwise} \end{Bmatrix},
\end{aligned} \tag{22}$$

where by a "special" equation we call the one for which the value of $S[j]$ is known.

Complexity $C^{\mathtt{B}}_{\mathtt{IR}}(L;a;q_{\mathtt{max}})$ This is the IR block where one equation (associated with time t) has $S_t[j_t]$ known. There could be three cases similar to $C_{\mathtt{MC}}$. However, these cases are not chosen by us likewise in MC, but rather one of them appears with some probability. The probability that the value z_t is in the clique of size $q+1$ is

$$\mathcal{P}_{\mathtt{A1}}(L;a;q_{\mathtt{max}};q) = \binom{a-1}{q} \frac{(N-L)\cdot M(N-L-1;a-q-1;q_{\mathtt{max}})}{M(N-L;a;q_{\mathtt{max}})}, \tag{23}$$

and the targeting complexity is

$$\begin{aligned}
C^{\mathtt{B}}_{\mathtt{IR}}(L;a;q_{\mathtt{max}}) = \sum_{q=0}^{q_{\mathtt{max}}-1} & \mathcal{P}_{\mathtt{A1}}(L;a;q_{\mathtt{max}};q) \times \Big[\\
& \underbrace{\frac{q}{N}}_{\substack{S^{-1}[z] \\ \text{is one} \\ \text{of the } q}} \cdot \underbrace{\frac{1}{N}}_{\substack{\text{No contra-} \\ \text{diction in} \\ \text{the clique} \\ \text{of size } q}} \cdot \mathcal{P}_c(L+1;q-1)\cdot C^{\mathtt{A0}}_{\mathtt{IR}}(L+1,q-1,a-q-1,q_{\mathtt{max}}) \\
& + \frac{a-q-1}{N}\cdot \mathcal{P}_c(L+1;q)\cdot C^{\mathtt{A1}}_{\mathtt{IR}}(L+1;q;a-q-1;q_{\mathtt{max}}) \\
& + \frac{N-L-a+1}{N}\cdot \mathcal{P}_c(L+1;q)\cdot C^{\mathtt{A0}}_{\mathtt{IR}}(L+1;q;a-q-1;q_{\mathtt{max}})\Big].
\end{aligned} \tag{24}$$

A.4 How to Apply the Complexities?

When the pattern is known and $\Delta_{\mathtt{thr}} \neq 0$, the complexity function should be applied at the point where the MC block is called. In this case $C_{\mathtt{MC}}(L; a; q_{\mathtt{max}})$ is added to the total complexity counter, where L and a are known, and $q_{\mathtt{max}}$ is the size of the maximum clique that had been previously found during the simulation.

When the pattern is unknown ($\Delta_{\mathtt{thr}} = 0$) but its parameters $d, w, l, b_\alpha, b_\beta, b_\gamma, b_\theta$ are given, the upper bound of the total complexity is calculated as

$$\begin{aligned} C_{\mathtt{Rand}} &< \mathcal{P}_c(d, b_\gamma) \cdot C^{\mathtt{AO}}_{\mathtt{IR}}(0; d + b_\gamma; w - l - b_\theta; w - l - b_\theta), && \text{for } \textit{random} \text{ keystream,} \\ C_{\mathtt{True}} &< C^{\mathtt{AO}*}_{\mathtt{IR}}(b_\gamma; d; w - l; 1), && \text{for } \textit{true} \text{ keystream,} \end{aligned} \tag{25}$$

where $C^{\mathtt{AO}*}_{\mathtt{IR}}$ is the same as $C^{\mathtt{AO}}_{\mathtt{IR}}$ except that the first call of the IR block may not have contradictions [5].

A.5 Restricted Verification Tests on Random Keystream

A set of patterns for restricted verification tests were chosen such that practical simulations of the attack would have as close conditions to the assumptions in Section A.2 as possible. We set $\Delta_{\mathtt{thr}} = 0, C_{\mathtt{K}}(L) = 0$, switch off the WE and GSi blocks, take patterns with $b_\alpha = b_\beta = 0$, and test them on a random keystream.

(Logarithms of the complexities)		Tests show that theoretical complexities behave adequately							Tests show that the real complexity depends on a certain pattern used					
	Pattern	$\mathcal{G}_2$	$\mathcal{G}_{3a}$	$\mathcal{G}_{4a}$	$\mathcal{G}_{4b}$	$\mathcal{G}_5$	$\mathcal{G}_6$	$\mathcal{G}_7$	$\mathcal{G}_{3b}$	$\mathcal{G}_{3c}$	$\mathcal{G}_{3d}$	$\mathcal{G}_{4c}$	$\mathcal{G}_{4d}$	$\mathcal{G}_{4e}$
	d	2	3	4	4	5	6	7	3	3	3	4	4	4
N	w	5	8	11	13	16	20	25	7	7	7	9	9	9
16	Pract	10.16	4.74	0.60	–	–	–	–	5.87	5.09	6.09	1.09	1.26	1.19
	Theor	9.76	4.65	0.98	–	–	–	–	5.96	5.96	5.96	2.14	2.14	2.14
30	Pract	19.90	24.22	21.22	17.90	8.71	1.84	–	22.69	22.73	22.90	22.50	22.87	22.27
	Theor	19.32	23.50	20.49	17.06	7.65	1.92	–	22.41	22.41	22.41	21.99	21.99	21.99
38	Pract	–	–	–	–	25.73	12.25	2.66	–	–	–	–	–	–
	Theor	–	–	–	–	24.78	11.54	2.59	–	–	–	–	–	–

Table 5. Results of restricted verification tests.

The results of the tests are given in Table 5. The first group of tests show that theoretical complexities behave adequately along with practical simulations for both small and large inputs. The second group of tests show that the actual complexity of the attack depends on a certain pattern, and it may vary.

[5] Brief boundings that need only d and w are $C^{\mathtt{AO}}_{\mathtt{IR}}(0; d; w; w)$ and $C^{\mathtt{AO}*}_{\mathtt{IR}}(0; d; w; 1)$.

A.6 Why Is Part-1 Needed?

Consider the pattern $A = \{0, 0, \{3, 1\}, \{1, 2\}\}$ and $N = 28$, $q_{\mathtt{max}} = 1$, the length of the window is $w = 5$. The probability of exactly one equation to be solved during the first iteration of the IR block is 0.3042, then a new value of $S[j]$ is received. In theory the probability of no contradiction would happen is $(N-L-l)/N \approx 0.928$, whereas in practice it is around 0.6, and this is a large deviation.

This simple example shows that no assumptions could cover all peculiarities of an actual pattern used. Therefore, when a precise pattern is given, it would be advised to run partial simulations of the attack in order to test top level branches of the recursion with the depth 1-3, since the case of the remaining subtrees becomes well compliant with the assumptions. This solution can attune theoretical complexity significantly in some cases.

A.7 Full Verification Tests on True Keystream

In order to verify reliability of complexity functions a set of full verification tests for three attack scenarios were carried out. For all scenarios $N = 64$, the patterns are $\mathcal{M}_8$, $\mathcal{M}_9$, and $\mathcal{M}_{10}$, and a true keystream is generated randomly. The four blocks in practice and the part with Knudsen's attack in theory are switched on.

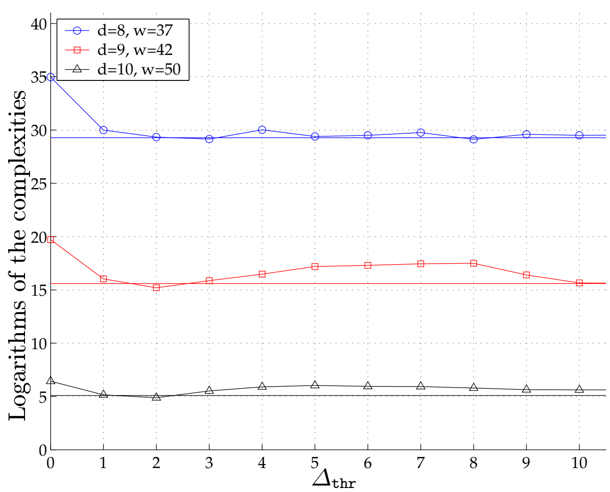

Fig. 9. Three patterns, true keystream, full attack, $N = 64$. The results of full verification tests of complexity functions of the new state recovering attack.

Figure 9 shows the results of the tests for the three scenarios. Real complexities received via simulations of the state recovering algorithm are horisontal lines, wherease the curves are theoretical upper bounds of these complexities for various $\Delta_{\mathtt{thr}}$, respectively. When $\Delta_{\mathtt{thr}} = 0$, points on the curves are *pattern independent* upper bounds.

B Patterns Used in This Paper

Reference Ref.	level of analysis A.l.	Pattern description i, j	P, V	order d	generative w	definitive l	α-predictive b_α	β-predictive b_β	θ-predictive b_θ	cumulative distance Π_θ	chain distance σ	intersection ξ	new guesses ψ	$\Pr\{E_{\text{int}}\}$ Int	$\Pr\{E_{\text{ext}}\}$ Ext
(a) Trade-off between w and l, the first level of analysis															
$\mathcal{X}_2$	1st	-3, 0	P ={-2, -1}, V ={0, 1}	2	4	2	2	0	0	0	–	–	–	N^{-3}	N^{-2}
$\mathcal{X}_3$	1st	-6, -7	P ={-5, -3, -2}, V ={3, 2, -1}	3	10	3	3	0	0	0	–	–	–	N^{-4}	N^{-3}
$\mathcal{X}_4$	1st	-10, -11	P ={-9, -7, -6, -2}, V ={3, 2, -4, -1}	4	14	3	3	0	0	0	–	–	–	N^{-5}	N^{-3}
$\mathcal{X}_5$	1st	-7, -2	P ={-6, -5, -3, -1, 1}, V ={-2, 4, 7, -1, 1}	5	16	5	4	0	0	0	–	–	–	N^{-6}	N^{-4}
$\mathcal{X}_6$	1st	-8, -10	P ={8, 9, -7, -5, -4, -3}, V ={2, -1, 4, 3, -2, 1}	6	23	4	4	0	1	2	–	–	–	$N^{-7}e^{-4/N}$	N^{-5}
$\mathcal{X}_7$	1st	-13,-2	P ={8, 9, -12, -11, -10, -9, -7} V ={2, -1, -2, 1, -5, 3, 4}	7	28	4	4	0	2	4	–	–	–	$N^{-8}e^{-8/N}$	N^{-6}
$\mathcal{X}_8$	1st	-18,-5	P ={8 9 -17 -16 -15 -14 -13 -9} V ={2, -1, -5, -2, 1, 4, 5, 3}	8	33	5	5	0	2	4	–	–	–	$N^{-9}e^{-8/N}$	N^{-7}
$\mathcal{X}_9$	1st	-20, -23	P ={0, 1, 5, -19, -17, -16, -15, -14, -7} V ={-5, 8, 3, 5, 4, -2, -1, 2, 1}	9	41	5	4	0	2	4	–	–	–	$N^{-10}e^{-8/N}$	N^{-6}
$\mathcal{X}_{10}$	1st	-25, -25	P ={6, 8, -24, -22, -20, -19, -18, -17, -3, -2 } V ={3, 4, 2, 8, -3, -2, 1, 7, 0, -5}	10	47	6	5	0	2	4	–	–	–	$N^{-11}e^{-8/N}$	N^{-7}
$\mathcal{X}_{11}$	1st	-37, -37	P ={-36, -35, -34, -33, -30, -29, -28, -15, -13, -10, 5} V ={10, -4, -1, 11, 3, -2, 1, 9, -3, -7, 2}	11	49	11	9	0	.	.	–	–	–	N^{-12}	N^{-9}
(b) Good detection through the second level of analysis															
$\mathcal{Y}_4$	1st	-7, -7	P ={-6, -5, -3, -1}, V ={3, 2, -1, 0}	4	9	4	4	0	.	.	–	–	–	N^{-5}	N^{-4}
	2nd	-2, -1	P' ={0, 2, -1}, V' ={0, -1, 2}	3	4	3	3	0	.	.	-4	3	0	$N^{-6}e^{-3}$	N^{-7}
$\mathcal{Y}_7$	1st	-24, -19	P ={1, -23, -22, -20, -18, -10, -3} V ={-3, -2, 4, 5, 1, 0, -1}	7	26	6	5	0	.	.	–	–	–	N^{-8}	N^{-5}
	2nd	-5, -2	P' ={0, 1, -4, -3, -2}, V' ={2, -1, 1, 0, -2}	5	6	5	5	0	.	.	-7	4	1	$N^{-10}e^{-4}$	N^{-10}
$\mathcal{Y}_8$	1st	-26, -27	P ={-25, -24, -23, -20, -19, -18, -16, -4} V ={5, 1, 4, -3, -1, 2, 3, -2}	8	29	6	4	1	0	0	–	–	–	N^{-9}	N^{-5}
	2nd	-7, 2	P' ={0, 3, -6, -5, -4, -3, -2}, V' ={-2, 3, 0, -1, 1, -3, 2}	7	10	7	7	0	0	0	-10	6	1	$N^{-12}e^{-6}$	N^{-12}

Table 6. Various patterns that were achieved by our simulations (part I).

Ref.	i, j	P, V	d	w	l	b_α	b_β	b_γ	b_θ	Π_θ
		(a) Maximum generative patterns ($w \to$ max)								
$\mathcal{M}_2$	0, -1	P ={1, 3}, V ={3, -1}	2	6	0	0	0	0	1	1
$\mathcal{M}_3$	0, -1	P ={1, 3, 4}, V={3, 2, -1}	3	10	3	0	1	2	0	0
$\mathcal{M}_4$	0, -2	P ={1, 3, 4, 5}, V={4, 3, -2, 1}	4	15	1	0	0	1	1	2
$\mathcal{M}_5$	0, -2	P ={1, 2, 4, 6, 8}, V={5, 2, -3, 6, -1}	5	21	0	0	0	0	0	0
$\mathcal{M}_6$	0, 0	P ={1, 2, 3, 4, 5, 20}, V={7, -1, 5, -3, 2, -9}	6	27	3	0	1	2	0	0
$\mathcal{M}_7$	0, 5	P ={1, 2, 4, 6, 8, 9, 16}, V={-2, 4, 7, 1, 3, -3, 8}	7	31	4	0	0	4	1	2
$\mathcal{M}_8$	0, 5	P ={1, 2, 4, 6, 14, 18, 19, 25} V ={-2, 4, 5, 1, 3, -3, 2, -1}	8	37	6	0	1	5	0	0
$\mathcal{M}_9$	0, 9	P ={1, 2, 3, 6, 7, 8, 11, 20, 24} V ={-4, -1, 10, 3, -2, 11, 1, 4, -6}	9	42	6	0	1	5	1	2
$\mathcal{M}_{10}$	0, 3	P ={1, 2, 3, 5, 8, 10, 18, 21, 22, 23} V ={1, 5, -3, 8, -7, 3, -2, -5, 9, -1}	10	50	4	1	1	2	1	2
$\mathcal{M}_{11}$	0, -1	P ={1, 2, 3, 4, 6, 9, 11, 13, 21, 30, 33} V ={6, 5, -3, 1, 4, -4, 7, -1, 2, -9, 8}	11	55	10	0	1	9	0	0
$\mathcal{M}_{12}$	0, 6	P ={1, 2, 3, 4, 5, 9, 15, 17, 34, 35, 43, 45 } V ={2, -2, 1, 12, -7, 7, 8, -3, 0, -5, 3, 4}	12	59	8	1	0	7	2	4
$\mathcal{M}_{13}$	0, 0	P ={1, 3, 5, 6, 7, 8, 22, 23, 31, 32, 34, 44, 52} V ={2, 8, -3, -2, 1, 7, 4, -9, 5, 10, -14, -5, 3}	13	68	9	0	2	7	2	4
$\mathcal{M}_{14}$	0, 15	P ={1, 2, 3, 4, 5, 11, 13, 30, 31, 39, 40, 42, 52, 60} V ={-7, -2, 1, 2, 7, 8, -3, 4, -9, 5, 10, -14, -5, 3}	14	76	10	0	2	8	2	4
		(b) Patterns with all $S_t[j_t]$ different to test complexity functions								
$\mathcal{G}_2$	0, 0	P ={3, 1}, V ={1, 2}	2	5	0	0	0	0	0	0
$\mathcal{G}_{3a}$	0, -2	P ={1, 3, 4}, V ={4, -1, 3}	3	8	0	0	0	0	0	0
$\mathcal{G}_{3b}$	0, -4	P ={2, 1, 3}, V ={1, 8, -7}	3	7	0	0	0	0	0	0
$\mathcal{G}_{3c}$	0, -3	P ={2, 1, 3}, V ={1, 7, -6}	3	7	0	0	0	0	0	0
$\mathcal{G}_{3d}$	0, 0	P ={3, 1, 2}, V ={3, 5, -1}	3	7	0	0	0	0	0	0
$\mathcal{G}_{4a}$	0, -4	P ={1, 3, 4, 5}, V ={6, -2, 1, 4}	4	11	0	0	0	0	0	0
$\mathcal{G}_{4b}$	0, 5	P ={1, 2, 4, 6}, V ={-2, 4, 5, 1}	4	13	0	0	0	0	0	0
$\mathcal{G}_{4c}$	0, -3	P ={2, 3, 1, 4}, V ={1, 3, 8, -10}	4	9	0	0	0	0	0	0
$\mathcal{G}_{4d}$	0, -1	P ={5, 3, 1, 2}, V ={1, 5, 7, -2}	4	9	0	0	0	0	0	0
$\mathcal{G}_{4e}$	0, 7	P ={4, 3, 5, 1}, V ={1, 9, -8, -5}	4	9	0	0	0	0	0	0
$\mathcal{G}_5$	0, -6	P ={1, 3, 4, 5, 8}, V ={8, -3, -1, 7, 5}	5	16	0	0	0	0	0	0
$\mathcal{G}_6$	0, -2	P ={2, 8, 1, 6, 5, 12}, V ={1, 2, 5, 7, -3, -1}	6	20	0	0	0	0	0	0
$\mathcal{G}_7$	0, -2	P ={2, 8, 21, 1, 6, 5, 12}, V ={1, 2, 4, 5, 7, -3, -1}	7	25	0	0	0	0	0	0
		(c) Patterns to support assumptions								
$\mathfrak{A}_1$	0, -10	P ={5, 2, 1, 4}, V ={3, 4, 9, -1}	4	9	0	0	0	0	0	0
$\mathfrak{A}_2$	0, -3	P ={2, 3, 1, 4}, V ={1, 3, 8, -10}	4	9	0	0	0	0	0	0
$\mathfrak{A}_3$	0, -1	P ={5, 3, 1, 2}, V ={1, 5, 7, -2}	4	9	0	0	0	0	0	0
$\mathfrak{A}_4$	0, 0	P ={3, 1, 6, 9}, V ={1, 2, 6, -5}	4	9	0	0	0	0	0	0
$\mathfrak{A}_5$	0, 7	P ={4, 3, 5, 1}, V ={1, 9,-8, -5}	4	9	0	0	0	0	0	0
$\mathfrak{A}_6$	0, 9	P ={2, 4, 1, 6}, V ={2, 8,-6, -1}	4	9	0	0	0	0	0	0
$\mathfrak{B}_1$	0, -1	P ={8, 1, 7, 3}, V ={1, 3,-9, -1}	4	9	0	0	0	0	1	1
$\mathfrak{B}_2$	0, 0	P ={3, 1, 9, 6}, V ={1, 2, 3, -8}	4	9	0	0	0	0	2	10
$\mathfrak{B}_3$	0, 0	P ={1, 3, 8, 5}, V ={2, 3,-6, -3}	4	9	0	0	0	0	2	11
$\mathfrak{B}_4$	0, 5	P ={4, 2, 8, 1}, V ={1, 4,-7, -2}	4	9	0	0	0	0	1	2
$\mathfrak{B}_5$	0, 7	P ={2, 3, 1, 8}, V ={1, 4,-3, -2}	4	9	0	0	0	0	2	4
$\mathfrak{B}_6$	0, 9	P ={2, 4, 3, 1}, V ={1, 4,-3, -2}	4	9	0	0	0	0	3	15
$\mathfrak{B}_7$	0, 10	P ={5, 3, 1, 2}, V ={1, 5,-4, -2}	4	9	0	0	0	0	2	11

Table 7. Various patterns that were achieved by our simulations (part II).