New State Recovery Attack on RC4

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, Dmitry.Khovratovich@uni.lu

Abstract. The stream cipher RC4 was designed by R. Rivest in 1987, and it is a widely deployed cipher. 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 recovery attack which accepts the keystream of a certain length, and recovers the internal state. For the reduced RC4-100, our attack has total complexity of around 2^{93} operations, whereas the best previous attack (from Knudsen et al.) needs 2^{236} of time.

The complexity of the attack applied to the original RC4-256 depends on the parameters of specific states (patterns), which are in turn hard to discover. Extrapolated parameters from smaller patterns give us the attack of complexity about 2^{241}, and it is much smaller than the complexity of the best known previous attack 2^{779}. The algorithm of the new attack was implemented and verified on small cases.

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

1 Introduction

RC4 [Sch96] is a stream cipher designed by Ron Rivest in 1987, and since then it has been implemented in many various software applications to ensure privacy in communication. It is one of the most widely deployed stream ciphers and 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 the 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 attacks impractical. RC4-256 uses a variable length key from 1 to 256 bytes for its initialisation.

The initialisation procedure of RC4 has been thoroughly analysed in a large number of various papers, see e.g. [MS01,Man01,PP04]. These results show that the initialisation of RC4 is weak, and the secret key can be recovered with a small

D. Wagner (Ed.): CRYPTO 2008, LNCS 5157, pp. 297–316, 2008.

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 has attracted many cryptanalysis efforts. In most analyses the scenario assumes that keystream of some length is given, and either a distinguishing ([Gol97, FM00, Max05, Man05]) or a state recovery ([KMP$^+$98]) attack is of interest. A *state recovery attack* can be used to determine the actual security level of a cipher, if the initial internal state is considered as a secret key. The first state recovery attack was proposed by Knudsen et al in 1998 [KMP$^+$98]. This had a computational complexity of 2^{779}. Some minor improvements were found in other literature, e.g. [MT98], but still, there is no attack even close to 2^{700}. One interesting attempt to improve the analysis was recently done in [Man05]. However, that attack is only a potential one [1], and the pretending time complexity claimed was around 2^{290}.

In this paper we propose a new state recovery attack on RC4-N. For the original design RC4-256 the total time complexity of the attack is less than 2^{579}, and under some realistic assumptions (see Section 6) a complexity would drop to 2^{241} (2^{272} under pessimistic extrapolations), requiring keystream of a similar length. This would mean that there is no additional gain in using a secret key longer than 30 bytes. We also show that in general if the secret key is of length N bits or longer the new attack is faster than exhaustive key search.

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 recovery algorithm is used together with 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 has been implemented and verified for **small values of** N, it has determined the correct internal state in *every* simulation run. The success rate of the full attack is shown to be at least 98%. For **large values of** N, where simulations were impossible, an upper bound for the average complexity of the attack is derived and calculated.

In the precomputation stage we search for a proper pattern to use in the attack. However, in this paper we skip a detailed analysis of that complexity since it is upper bounded by the time needed for the main stage of the attack (see Appendix B).

This paper is organized as follows. In Section 2 the new iterative *state recovery algorithm* is described in detail. Afterwards, Section 3 introduces various properties of a pattern that are needed for the recovering algorithm. An effective searching algorithm to find such patterns is also proposed in Appendix B (due to the page limitation and clarity of presentation). Section 4 describes several

[1] Mantin detects a large number of bytes of the state, and then applies Knudsen's attack given those bytes. However, this would reduce the complexity only if the knowns were located in a short window all together while this is not the case. This fact is confirmed in [Man05] (Section "State Recovery Attack").

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 recovery algorithm and derivation of its complexity functions are performed in the full version of this paper [MK08]. All pieces of the attack are then 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 $\mathbf{z} = (z_1, z_2, \ldots)$, where z_i is a value $0 \leq z_i < N$. In all tables probabilities and complexities will be given in a logarithmical form with base 2.

1.2 Description of the Keystream Generator RC4-N

The new attack targets the keystream generation phase of RC4 and, thus, the initialisation procedure will not be described. We refer to, e.g., [Sch96] for a full description of RC4. 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 $0 \ldots N-1$
$S[\cdot]$ is initialised with the secret key
The keystream generator RC4-$N$
  $i = j = 0$
  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 Recovery 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:

$$j_t,\ S_t[i_t],\ S_t[j_t],\ S_t^{-1}[z_t]. \qquad (1)$$

One can simply simulate the pseudo random generation algorithm and, when necessary, guess these unknown values in order to continue the simulation. The recursion steps backward when a contradiction is reached due to previously wrong guesses. Additionally, it can be assumed that some k values are a priori 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 of [Man01]. As an example, the complete state recovery attack on RC4-256 would require time around 2^{779}.

2.2 Our Algorithm for State Recovery

In this section we propose an improved version of the state recovery algorithm. Assume that, 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

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

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

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

where only *two* variables are unknown,

$$S_k^{-1}[z_k], \quad S_k[j_k], \qquad (4)$$

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. For simplicity, let us set $t_0 = 0$.

Our new state recovery algorithm is a recursive algorithm, 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.

Now we give a more detailed description of the different parts of the algorithm.

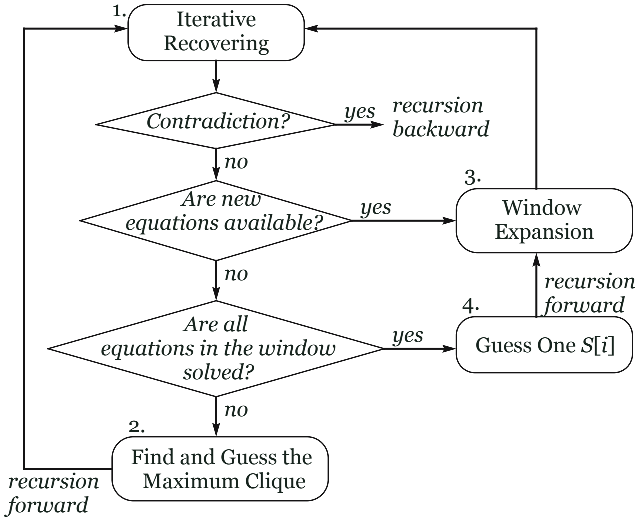

Fig. 2. New state recovery algorithm

Iterative Recovering (IR) Block. The *Iterative Recovering* block receives a number a of *active* equations (not yet processed) in the window of length w as input, and tries to derive the values of $S_t[j_t]$s and $S_t^{-1}[z_t]$s. To do that, the IR block goes through two steps iteratively, until no more new derivations are 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 finds a contradiction the recursion steps backward. The two steps are as follows.

A. Assume that, 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 using (3).
A contradiction arises if (a) $S_t[j_t]$ is already allocated and it is not equal to the derived value; (b) the derived value already exists in some other cell.

B. Already 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.

Find and Guess the Maximum Clique (MC) Block. If no more active equations can be explicitely be solved, $S_t^{-1}[z_t]$ for one t has to be guessed. The *Find and Guess the Maximum Clique* block analyses given active equations, and chooses the element that gives the maximum number of new derivations in consecutive recursive calls of the IR block. This element is then guessed.

The analysis is very simple. Let a active equations be vertices v_t in a graph representation. Two vertices $v_{t'}$ and $v_{t''}$ are connected if $z_{t'} = z_{t''}$ and/or $S_{t'}[j_{t'}]$

and $S_{t''}[j_{t''}]$ *refer* (like pointers) to the same cell of the ground state. Guessing any unknown variable in any connected subgraph solves all equations involved in that subgraph. Therefore, let us call these subgraphs *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.

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 the beginning or in the end 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 and 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.

Appendix A provides an example that shows the steps of the outlined algorithm.

3 Precomputations: Finding Good Patterns

The algorithm presented in the previous section is used in the full state recovery attack as a part of it. Every time when the algorithm is running at some point of the keystream, its effectiveness depends on certain properties of the current internal state. Although these properties are not visible for the intruder, she may have a good guess about places in the keystream where the internal state has good properties (see Section 4), and apply the state recovery algorithm only at those places.

In this section we will define patterns (see Definition 1), they determine huge sets of internal states with common properties. If, for instance, a pattern has a large window then this certainly helps decreasing the complexity of the algorithm. However, the probability that the internal state is compliant with a certain pattern decreases with the number of conditions put on the pattern.

In this section we discuss properties of patterns that influence on the complexity of the attack, and also study their availability. We have also developed an efficient algorithm for finding these paterns, and it is located in Appendix B.

3.1 Generative States

Let us start with several definitions, some of which were previously defined in [MS01, Man01, Man05].

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 4 in Appendix A 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. Thus, 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 to derive w equations of the form* (3)*, i.e., consecutive $w+1$ values of j*s *are known.* □

Table 1 demonstrates a 4-order 7-generative pattern A={-7,-8,{-6, -5, -4, 0}, {6, -1, 2, -2}}, that supports the above definitions. Eight equations involve symbols of the keystream $z_{t+1}, \ldots, z_{t+8}$ associated with a certain time t. We say that 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.

3.2 Availability

We have done a set of simulations in order to find *maximum w-generative d*-order patterns, denoted by M_d. The results are given in Table 7(a) in Appendix C.

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

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

	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

Searching for a high order pattern is a challenging task since the computational complexity grows exponentially with d. The best result achieved in our work is a 14-order 76-generative pattern M_{14}.

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

Conjecture 1. The rate of $\frac{w_{\mathtt{max}}}{d} \approx 6$ as $d \to \infty$.[2] □

That conjecture allows us to make a prediction about certain parameters for patterns with large d. These could not be found due to a very high precomputation complexity, but they are needed to analyse the attack for large N ($N = 128 \dots 256$ in Table 3). However, given those parameters, d and w, we can derive theoretical complexities of the attack on average [3]. This has been done in [MK08]. An efficient search algorithm for patterns with desired properties is given in Appendix B.

4 Detection of Patterns in the Keystream

In the previous section we have studied properties of a pattern that are desirable for the state recovery algorithm to work fast and efficient. We have also shown (in Appendix B) 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 the detection is very good, then the state recovery algorithm might only have to be executed once, at the right location in 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. On the other hand, a good detection may require a deep analysis of the keystream, where specific properties of the pattern can be used efficiently.

[2] Indeed, the "jump" of $w_{\mathtt{max}}$ as d increments by one is the sequence Γ={4, 5, 6, 6, 4, 6, 5, 8, 5, 6, 7, 8, ...}. Obviously, for small d this "jump" is small, and it is notable that the "jump" increases for larger d. In our simulations heuristics were used (see Appendix B) when searching patterns for $d \geq 6$. This means that our "jumps" in the sequence Γ could possibly be larger if an optimal searching technique is applied, since our heuristic cannot guarantee that we get a pattern with the longest window. This suggests that the ratio $w \to 6d$ as $d \to \infty$ seems quite a fair conjecture.

[3] Because the relation $w = 6d + \lambda$ is a subject of discussions, we show in Table 4 that even more pessimistic conjectures do not affect the total complexity very much.

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 occurs, there is an *external event* $E_{\mathtt{ext}}$ observed in the keystream, which is associated with the pattern A, i.e., $\Pr\{E_{\mathtt{ext}}|E_{\mathtt{int}}\} = 1$. Applying Bayes' law we can derive the *detection probability* $\mathcal{P}_{\mathtt{det}}$ of the pattern A in the keystream as

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

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.* □

It means that in l equations $S[i] + S[j]$ are known. If, additionally, $z' = S[S[i] + S[j]]$ is also known, then the correct value of $z_t = z'$ at the right position 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, a large l can be achieved via a slight reduction of the parameter w. This leads us to one more conjecture.

Conjecture 2. For any d and $w = w_{\mathtt{max}} - \lambda$ there exist a pattern with $l = d$, where λ is relatively small [4]. □

In the following definition we introduce other properties of a pattern that are important for its good detection via the keystream.

Definition 4 ($b_\alpha, b_\beta, b_\gamma$-$^{\alpha,\beta,\gamma}$predictive pattern). *Let us have an l-definitive pattern A and 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 $S[S[i] + S[j]]$.* □

[4] Table 6(a) in Appendix C contains patterns Xs with $l = d$ where w is still large, which supports the above conjecture. Indeed, Table 5 in Appendix B shows how the number of available patterns grows when relaxing the condition put on w. I.e., a slight reduction of w increases the chance of finding a pattern with $d = l$. This makes the conjecture fair.

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 and 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 time instances t_1 and t_2 one pair of these equations be such that the $S[i]$ values and the $S[j]$ pointers 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 $64^{-5} \div (64^{-2} \cdot 60 \cdot 59/64^2) \approx 64^{-2.96}$[5].

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 j is not detectable), whereas with the second level of analysis it can be 1. Let us introduce a technique that we call a *chain of patterns*.

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

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 is 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's 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 B, and after $\sigma = 4$ clockings a new pattern B ($w_b = 34$) can appear if $j_{t+34} = j_b$. The probability that the chain $A \to B$ appears is $N^{-9} \cdot N^{-4}$, multiplied by the probability that 5 elements from A' stay at the same locations during the next 4 clockings. This 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 Y_4 in Table 6 in Appendix C), confusing the equation (8). This happens since $\Pr\{E_{\mathtt{ext}}\}$ is calculated assuming that the keystream is random. However, in 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 the case when $\Pr\{E_{\mathtt{ext}}\} < \Pr\{E_{\mathtt{int}}\}$ we simply assume that the detection probability is 1.

Table 6 in Appendix C presents a few 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 d were given. However, we believe that for large d it is possible to detect such patterns with a high probability, up to 1, applying the two proposed levels of analysis.

5 Complete State Recovery Attack on RC4

5.1 Attack Scenario and Total Complexity

Recall pattern detection techniques from Section 4. In the attack scenario an adversary analyses the keystream at every time t, and applies the state recovery 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*

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

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 requires. 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. In this case the *success rate* of the attack can be determined.

Figure 3 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.

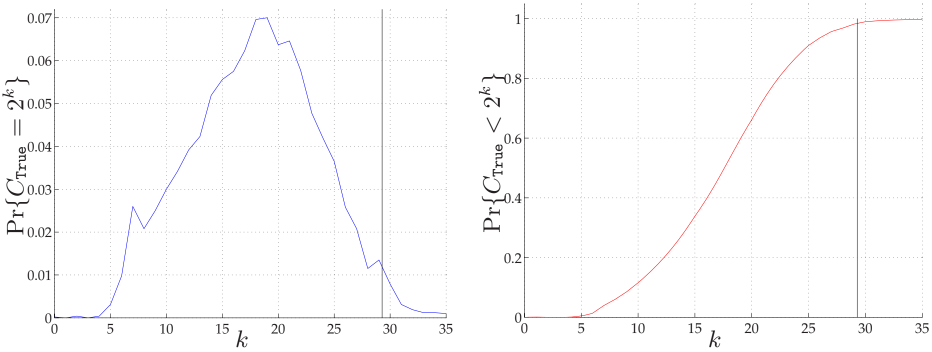

Fig. 3. Probability density (left) and cumulative (right) functions of the time $C_{\mathtt{True}}$ in logarithmical form ($k = \log_2 C_{\mathtt{True}}$). The scenario is $N = 64, \mathrm{M}_8$ and 2000 samples.

The plots in Figure 3 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 attack is still very high. I.e., the state recovery algorithm on RC4-64 can be done in time 2^{15} with success probability 35%! If a similar situation happens for large N (e.g., $N = 256$), 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 3 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 X_9 is used. This is much faster than, for example, Knudsen's attack, which 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* 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%.

Table 3 also contains intermediate probabilities and complexities for 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 state of RC4 was successfully recovered for every randomly generated keystream compliant with the corresponding pattern.

For larger N, patterns of a high order are needed to receive an attack of low complexity. The largest pattern that we could find in this work is M_{14}, and this 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 $w = 6d - 6$. 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 recovery attack known at the moment.

Since Conjecture 1 is discussible, we show in Table 4 that even pessimistic relations between w and d do not increase the attack complexity of approximated scenarious ($\star$) significantly. 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 3.

As the last point of the discussions we note that the key recovery attack can be easily converted from a state recovery attack. There are several papers dealing with recovering the secret key from a known internal state [MS01,Man01,

Table 3. Simulation results and comparisons with previous attacks

	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 ($\star$ – are hypothetical cases)												
	Pattern	$\mathcal{M}_8$	$\mathcal{Y}_8$	$\mathcal{X}_9$	$\mathcal{X}_{11}$	$\mathcal{M}_{13}$	$\mathcal{M}_{14}$	$\star$	$\mathcal{M}_{14}$	$\star$	$\mathcal{M}_{14}$	$\star$	$\mathcal{M}_{14}$	$\star$
	d	8	8	9	11	13	14	15	14	18	14	23	14	29
	w	37	29	41	49	68	76	84	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 recovery 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	65.8	**60.0**	79.7	**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. Complexities of the attack on RC4-256 for various relations $w = \xi d + \lambda$. All scenarious show much better attack complexity than the best previous one 2^{779}.

	optimistic	realistic		pessimistic	
$N = 256$	$w = 6.5d - 17$	$w = 6d - 6$	$w = 6d - 12$	$w = 5d$	$w = 4d$
d, w	$d = 29, w = 171$	$d = 29, w = 168$	$d = 30, w = 168$	$d = 33, w = 165$	$d = 39, w = 156$
keystream	240.0	240.0	248.0	272.0	320.0
time	224.9	241.1	243.3	265.9	327.1

BC08]. However, this part works much faster than currently known state recovery attacks, and, therefore, we just refer to these papers without giving details.

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 recovery 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 a more accurate estimation of the total complexity, and it might improve the complexities in Table 3 significantly. Another interesting problem is to determine the density function of the recovering algorithm, likewise in Figure 3. This may allow us to decrease the complexity in square root times, maintaining a high success rate.

Other open problems. The search for patterns of a higher order with long windows is another challenging 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, another interesting question is whether or not the second pattern can also be used in the recovering algorithm.

Acknowledgements

We thank Martin Hell, Lars Knudsen, Matt Robshaw and also anonymous reviewers for their valuable comments and efforts which helped us to improve this paper significantly. This work was partly supported by University of Luxembourg and Ericsson AB.

References

[BC08] Biham, E., Carmeli, Y.: Efficient reconstruction of rc4 keys from internal states. In: Fast Software Encryption 2008. Lecture Notes in Computer Science. Springer, Heidelberg (to appear, 2008)

[FM00] Fluhrer, S.R., McGrew, D.A.: Statistical analysis of the alleged RC4 keystream generator. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 19–30. Springer, Heidelberg (2001)

[Gol97] Golić, J.D.: Linear statistical weakness of alleged RC4 keystream generator. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 226–238. Springer, Heidelberg (1997)

[KMP+98] Knudsen, L.R., Meier, W., Preneel, B., Rijmen, V., Verdoolaege, S.: Analysis methods for (alleged) RC4. In: Ohta, K., Pei, D. (eds.) ASIACRYPT 1998. LNCS, vol. 1514, pp. 327–341. Springer, Heidelberg (1998)

[Man01] Mantin, I.: 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] Mantin, I.: Predicting and distinguishing attacks on RC4 keystream generator. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 491–506. Springer, Heidelberg (2005)

[Max05] Maximov, A.: Two linear distinguishing attacks on VMPC and RC4A and weakness of RC4 family of stream ciphers. In: Gilbert, H., Handschuh, H. (eds.) FSE 2005. LNCS, vol. 3557, pp. 342–358. Springer, Heidelberg (2005)

[MK08] Maximov, A., Khovratovich, D.: New state recovery attack on RC4 (accessed May 27, 2008) (2008), *http://eprint.iacr.org/2008/017*

[MS01] Mantin, I., Shamir, A.: Practical attack on broadcast RC4. In: Matsui, M. (ed.) FSE 2001. LNCS, vol. 2355, pp. 152–164. Springer, Heidelberg (2002)

[MT98] Mister, S., Tavares, S.E.: Cryptanalysis of RC4-like ciphers. In: Tavares, S., Meijer, H. (eds.) SAC 1998. LNCS, vol. 1556, pp. 131–143. Springer, Heidelberg (1999)

[PP04] Paul, S., Preneel, B.: A new weakness in the RC4 keystream generator and an approach to improve the security of the cipher. In: Roy, B., Meier, W. (eds.) FSE 2004. LNCS, vol. 3017, pp. 245–259. Springer, Heidelberg (2004)

[Sch96] Schneier, B.: Applied Cryptography: Protocols, Algorithms, and Source Code in C, 2nd edn. John Wiley&Sons, New York (1996)

A Example Support for the State Recovery Algorithm

Figure 4 illustrates an example of the process of the IR block. In the 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 rightmost column of the table.

The first iteration, in Figure 4(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 4(c). Finally, after the step (e) four additional cells of the state S were derived with probability 1.

When the IR block is processed, the input to the MC block is the maximum clique of size 4 equations with 5 unknowns, shown in Figure 4(f). It means that guessing only one unknown determines four other ones. Furthermore, the space of possible guesses is significantly reduced due to the higher probability of a contradiction to occur.

		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	?																							

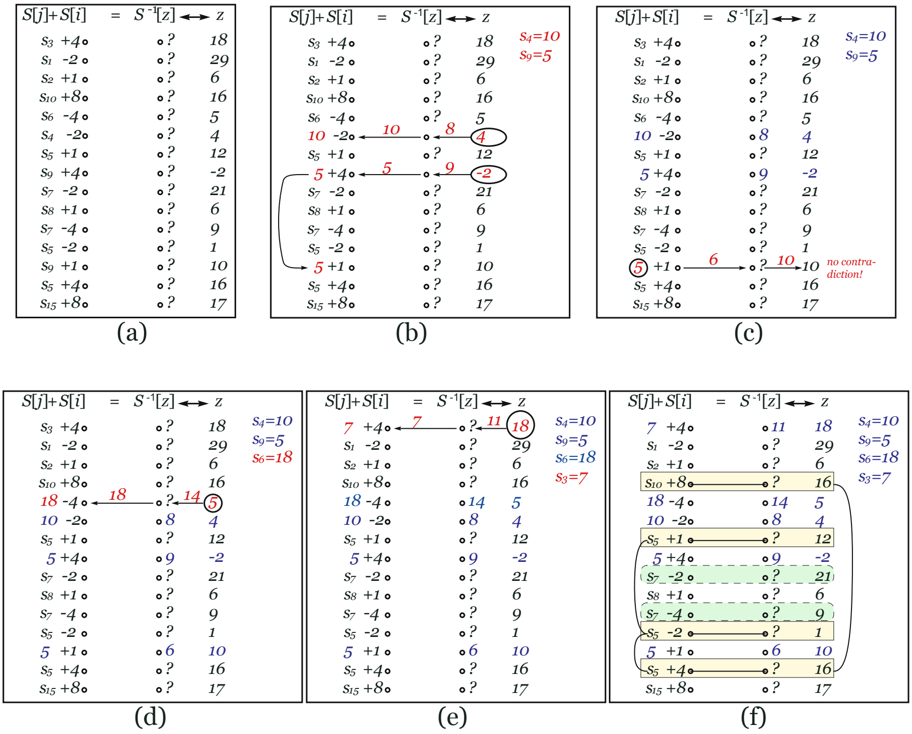

Fig. 4. Example of the iterative reconstruction process

B Searching Technique

Since the search space for a d-order pattern grows exponentially with d, only patterns of order $d \leq 6$ were analysed before in various literature, e.g., in [Man05]. In this section we suggest a few techniques that accelerate this search significantly, and allow to search and analyse patterns of order up to $d \leq 15$, approximately, on a usual desktop PC.

First, we need to make some observations on the construction of patterns. Afterwards, several ideas based on the observation for improving the algorithm follow.

All "good" patterns found have Vs with values from a short interval $I_\delta = [-\delta \ldots + \delta]$, where $\delta \approx 10 \ldots 25$ is quite conservative. From this we make the following conjecture.

Conjecture 3. 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 $\delta \ll N$. □

This conjecture will be used as the basis for a *significant* improvement in the *searching technique* of such patterns.

Table 5 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 be exponentially increased by letting w be slightly less than $w_{\mathtt{max}}$. This approach can help finding patterns with additional properties which are introduced in Section 4.

The first idea is to set $i = 0$ due to (6), and for the remaining variables only a small set of values I_δ for some δ should be tested due to Conjecture 3.

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

The diagram of a 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 as $d = d_{\mathtt{fwd}} + d_{\mathtt{back}}$ and then start the algorithm in Figure 5 allowing to allocate exactly $d_{\mathtt{fwd}}$ cells of S. At the point $(*)$ the current length of the window w is compared with some threshold $w_{\mathtt{thr}}$. If $w \geq w_{\mathtt{thr}}$, then a similar recursive algorithm starts, but it goes backward and allocates remaining $d_{\mathtt{back}}$ cells of S. This double-recursion results in a pattern with w *likely* to be close to the maximum possible length of the window.

Searching of a d-order pattern is a precomputation stage of the attack.

Theorem 2. *The complexity of the precomputation stage is less than the total complexity of the attack.*

Table 5. The number of different constraints for specific d and w, when $\delta = 15$

d	The number of patterns A_d when $\delta = 15$.									
↓ $w \rightarrow$	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

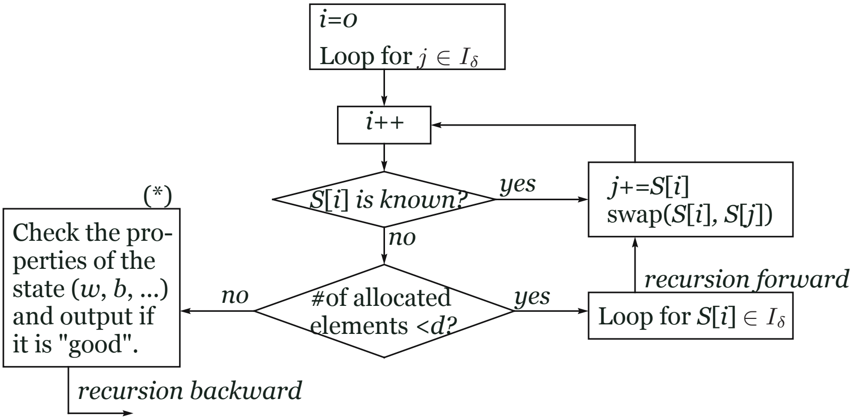
i=o
Loop for $j \in I_\delta$
i++
S[i] is known?
yes
j+=S[i]
swap(S[i], S[j])
no
(*)
Check the properties of the state (w, b, ...) and output if it is "good".
no
#of allocated elements <d?
yes
recursion forward
Loop for $S[i] \in I_\delta$
recursion backward

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

Proof. Assume we are interested a d-order pattern. To start with, one should loop j of N values. Afterwards, the algorithm tries to allocate the first value in $V[]$ at some first location in $P[]$, which is another inner loop of N values, and so on. At the end we got $d+1$ inner loops, each of N values. Thus, the complexity of this non-heuristic and non-optimized searching algorithm is $O(N^{d+1})$. The attack requires a keystream of the same size, thus, it proofs the statement. □

C Patterns Used in This Paper

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

Reference	level of analysis	Pattern description		order	generative	definitive	α-predictive	β-predictive	θ-predictive	cumulative distance	chain distance	intersection	new guesses	$\Pr\{E_{\text{int}}\}$	$\Pr\{E_{\text{ext}}\}$
Ref.	A.l.	i, j	P, V	d	w	l	b_α	b_β	b_θ	Π_θ	σ	ξ	ψ	Int	Ext
(a) Trade-off between w and l, the first level of analysis															
$\mathcal{X}_8$	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 7. Various patterns that were achieved by our simulations (part II)

Ref.	i, j	P, V	d	w	l	b_α	b_β	b_γ	b_θ	Π_θ
$\mathcal{M}_2$	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