Cryptanalysis of Achterbahn

Thomas Johansson[1], Willi Meier[2] *, and Frédéric Muller[3]

[1] Department of Information Technology, Lund University
P.O. Box 118, 221 00 Lund, Sweden
thomas@it.lth.se
[2] FH Aargau, 5210 Windisch, Switzerland
w.meier@fh-aargau.ch
[3] HSBC-France
Frederic.Muller@m4x.org

Abstract. We present several attacks against the Achterbahn stream cipher, which was proposed to the eSTREAM competition. We can break the reduced and the full version with complexity of 2^{55} and 2^{61} steps. Extensions of our attacks are also described to break modified versions of the Achterbahn stream cipher, which were proposed following the publication of preliminary cryptanalysis results.
These attacks highlight some problems in the design principle of Achterbahn, *i.e.*, combining the outputs of several nonlinear (but small) shift registers using a nonlinear (but rather sparse) output function.

1 Introduction

The European project ECRYPT recently decided to launch a competition to identify new stream ciphers that might be suitable for widespread adoption. This project is called eSTREAM [3] and received 35 submissions, some of which have already been broken.

Among these new algorithms, a challenging new design is Achterbahn [5]. It is a relatively simple, hardware-oriented stream cipher, using a secret key of 80 bits. In this paper, we present several attacks which break the cipher faster than a brute force attack. Our results provide new directions to break stream ciphers built by combination of several small, but nonlinear shift registers, like Achterbahn.

2 Description of Achterbahn

2.1 General structure

Achterbahn uses 8 small non-linear registers, denoted by $R_1, \ldots, R_8$. Their size ranges from 22 to 31 bits (see Table 1). The total size of the internal state is 211 bits. At the t-th clock cycle, each register produces one output bit, denoted

* The second author is supported by Hasler Foundation www.haslerfoundation.ch under project number 2005.

Register	Length
R_1	22
R_2	23
R_3	25
R_4	26
R_5	27
R_6	28
R_7	29
R_8	31

Table 1. Length of non-linear registers in Achterbahn

respectively by $y_1(t), \ldots, y_8(t)$. Then, the t-th output bit $z(t)$ of the stream cipher Achterbahn is produced by the filtering function F as

$$\begin{aligned} z(t) &= F(y_1(t), y_2(t), y_3(t), y_4(t), y_5(t), y_6(t), y_7(t), y_8(t)) \\ &= y_1(t) \oplus y_2(t) \oplus y_3(t) \oplus y_4(t) \oplus \\ &\quad y_5(t)y_7(t) \oplus y_6(t)y_7(t) \oplus y_6(t)y_8(t) \oplus y_5(t)y_6(t)y_7(t) \oplus y_6(t)y_7(t)y_8(t). \end{aligned}$$

We can observe that F is a sparse polynomial of degree 3. There are only 3 monomials of degree 2 and 2 monomials of degree 3. In the full version of Achterbahn, the input of F is not directly the output of each register, but a key-dependent combination of several consecutive outputs[1]. In the reduced version of Achterbahn, the input of F is directly the output of each register.

Each register is clocked similarly to a Linear Feedback Shift Register (LFSR), except that the feedback bit is not a linear function, but a polynomial of degree 4. Details of this clocking are not relevant in our attack. We refer to the original description of Achterbahn for more details [5].

2.2 Initialization

The internal state of Achterbahn is initialized from a secret key K of size 80 bits and from an initialization vector IV of length 80 bits.

First, the state of each register is loaded with a certain number of key bits (this number depends on the register length). Then, the rest of the key, followed by the IV, is introduced sequentially in each register. More precisely, this introduction consists simply in XORing the auxiliary input to the feedback bit during the register update. At some point, one bit in the register is forced to 1 to prevent the all-zero state. Before the encryption starts, several extra clockings are applied for diffusion purpose.

[1] The number of consecutive outputs involved in this linear combination varies from 6 for R_1 to 10 for R_8.

2.3 Evolutions of Achterbahn

In September 2005, some preliminary cryptanalysis results were announced on the eSTREAM website [6]. These results allow to break the reduced version of Achterbahn with 2^{56} computation steps and the full version with complexity of 2^{73} computation steps.

After the publication of these results, the designers of Achterbahn proposed to modify the output filter F of Achterbahn in order to strengthen the cipher [4]. This is a natural idea, given the nature of the published attacks. The first suggestion, that we will refer to as **Achterbahn-v2** in this paper, uses a new combining function F' instead of F, where

$$F'(y_1(t),\dots,y_8(t)) = F(y_1(t),\dots,y_8(t)) \oplus y_5(t)y_6(t) \oplus y_5(t)y_8(t) \oplus y_7(t)y_8(t).$$

Another alternative suggested in [4] is to replace F by F'' defined as

$$F''(y_1(t),\dots,y_8(t)) = y_1(t) \oplus y_2(t) \oplus y_3(t) \oplus \sum_{4\le i<j\le 8} y_i(t)y_j(t) \oplus \sum_{4\le i<j<k\le 8} y_i(t)y_j(t)y_k(t) \oplus \sum_{4\le i<j<k<l\le 8} y_i(t)y_j(t)y_k(t)y_l(t).$$

We refer to **Achterbahn-v3** for the cipher instantiated with F''.

3 Weaknesses of Achterbahn's design

3.1 General observations about the design

Combination of several small Linear Feedback Shift Registers (LFSR) is a well-known method for building stream ciphers. The output of the registers are generally combined with a function F, in order to produce one keystream bit (see Figure 1). A popular example is the algorithm E0 [1], which is used in the Bluetooth technology[2]. Unfortunately such constructions have some problems, that

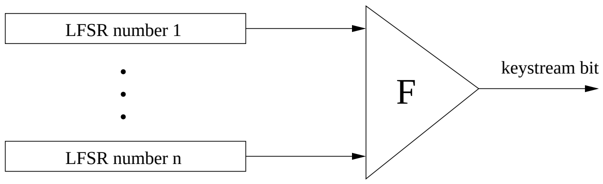

Fig. 1. Stream Cipher built by Combination of LFSR's

originate from the linearity of the LFSR's. For instance, correlation attacks [8,

[2] E0 has the particularity that the function F uses a small auxiliary memory.

9] exploit linear approximations of the function F to attack the whole stream cipher. Another method is algebraic attacks [2] that take advantage of low degree polynomial equations satisfied by F.

Criteria that should be satisfied by the boolean function F, in order to counter such attacks have been widely studied. However there appears to be limitations that cannot easily be removed. To improve the designs, it is often suggested to replace linear registers by nonlinear registers. This idea is the bottomline of Achterbahn's design.

3.2 Linear Complexity of Achterbahn

If the linear registers of Figure 1 are replaced by nonlinear registers, one may expect to counter many problems arising from the linearity of LFSR's. A usual tool to analyze such constructions is the **linear complexity**. For a binary sequence, it is defined as the length of the shortest LFSR that could generate the sequence.

For a LFSR of length n bits, the linear complexity of its output sequence is $L = n$, provided its feedback polynomial is properly chosen. For a nonlinear register, it is not always easy to compute the linear complexity of its output sequence, but clearly it cannot exceed its period. In the case of Achterbahn, the keystream bit b is computed by

$$b = F(y_1, \ldots, y_8).$$

Then, it is well-known that the linear complexity of the keystream sequence is at most

$$L = F(L_1, \ldots, L_8),$$

where L_i denotes the linear complexity of each single register and F is now seen as a polynomial on the integers, with its coefficients $\in \{0, 1\}$. This observation shows that **it would be insecure to combine the small nonlinear registers using a linear function**. Indeed, in this case, the linear complexity L of Achterbahn would be bounded by 8×2^{31} since 31 is the length of the largest register.

For Achterbahn, F is not linear, but its algebraic degree is 3. The original paper [5] does not contain an exact proof of the linear complexity of the 8 nonlinear registers, but it is reasonable to assume that $L_i \simeq 2^{n_i}$ where n_i denotes the length of register R_i. With this assumption, the linear complexity of Achterbahn's outputs is :

$$L \leq 2^{28} \times 2^{29} \times 2^{31} = 2^{88}.$$

If we apply the Berlekamp-Massey algorithm [7], we can expect to distinguish this sequence if we analyze 2^{89} known output bits. Since the running time of Berlekamp-Massey is about L^2, this attack is way above the complexity of a brute-force attack.

3.3 Ideas for improvement

These observations about the linear complexity were taken into account by the designers of Achterbahn (see page 20 of [5]). However, we should also consider that several refinements are possible :

- The output function is **sparse**. Indeed $z(t)$ is computed by a simple filter, which is almost linear. For instance, when $y_6(t) = 0$, only one nonlinear term remains. If $y_5(t)$ is also equal to 0, the output function becomes purely linear.
- Each single register has a **small period**. This is unavoidable due to the small size of each register (31 bits for the largest one, R_8).
- Each register is **autonomous**. Therefore when we guess its initial state, we know its content at all stages of the encryption.

Our idea is to guess the initial state of two registers (R_5 and R_6). Then we **select** particular positions in the output sequence, for which

$$y_5 = y_6 = 0.$$

All nonlinear terms in F cancel out, so the linear complexity of this subsequence is much smaller than for the whole Achterbahn. Finally, we test if several **parity checks**, resulting from the low linear complexity are satisfied or not. Hence, we can determine when the initial guess on R_5 and R_6 is correct.

Several tricks are needed in order for the attack to work properly. In particular, it is important to find low-weight parity checks. The details of this attack are given in the next section. Attacks in the same vein can also be mounted in the case of Achterbahn-v2 and Achterbahn-v3.

4 Cryptanalysis of Reduced Achterbahn

4.1 Preliminary

Our starting point is to observe that when $y_5(t) = 0$ and $y_6(t) = 0$, the output function becomes purely linear, so

$$z(t) = l(t) = y_1(t) \oplus y_2(t) \oplus y_3(t) \oplus y_4(t).$$

Although its period is rather large, $l(t)$ has a very **low linear complexity** L as pointed out in Section 3.2. Indeed, L is bounded by

$$L \leq 2^{n_1} + 2^{n_2} + 2^{n_3} + 2^{n_4} \simeq 2^{26}.$$

By definition, l can be generated by a LFSR of length L, so it will satisfy some **parity checks** involving L consecutive bits at most. Actually, it can be demonstrated that **sparse parity checks** are satisfied, which will prove to be crucial in the rest of our attack.

4.2 Construction of Sparse Parity Checks

We denote by T_i the period of register R_i. From [5], we can see that

$$\begin{aligned} T_1 &= 2^{22} - 1, \\ T_2 &= 2^{23} - 1, \\ T_3 &= 2^{25} - 1, \\ T_4 &= 2^{26} - 1. \end{aligned}$$

Let

$$ll(t) = l(t) \oplus l(t + T_1).$$

Because the period of the first register is T_1, this expression does not contain any term in y_1. Similarly, define

$$\begin{aligned} lll(t) &= ll(t) \oplus ll(t + T_2), \\ llll(t) &= lll(t) \oplus lll(t + T_3). \end{aligned}$$

Here $llll(t)$ contains no term in y_2 or y_3, so it is a combination of bits coming from the register R_4 only. Thus it satisfies

$$llll(t) = llll(t + T_4).$$

In other terms, we have the following relation on the bits $l(i)$,

$$\begin{aligned} 0 = {} & l(t) + l(t+T_1) + l(t+T_2) + l(t+T_3) + l(t+T_4) \\ & + l(t+T_1+T_2) + l(t+T_1+T_3) + l(t+T_1+T_4) \\ & + l(t+T_2+T_3) + l(t+T_2+T_4) + l(t+T_3+T_4) \\ & + l(t+T_1+T_2+T_3) + l(t+T_1+T_2+T_4) + l(t+T_1+T_3+T_4) \\ & + l(t+T_2+T_3+T_4) + l(t+T_1+T_2+T_3+T_4). \end{aligned}$$

This is the basic **parity check** on $l(t)$ that we will use in our attack. We can observe that it is the XOR of 16 different bits from the sequence $l(i)$. They all belong to a time interval of length

$$T_{max} = T_1 + T_2 + T_3 + T_4 = 113246204 \simeq 2^{26.75}.$$

Such parity checks are satisfied by the keystream sequence, under certain constraints on the outputs of the registers R_5 and R_6 (several bits $y_5(i)$ and $y_6(i)$ must be equal to 0).

We split the attack in two phases. First, we **precompute** particular states of R_5 and R_6 for which $z(t) = l(t)$. Then we look at a given keystream sequence and test when the parity check is satisfied. This information is used to **identify** one of the precomputed states of R_5 and R_6.

4.3 Precomputation

The goal of the precomputation step is to identify particular state values of R_5 and R_6 for which the parity checks will be satisfied. For that, we need $y_5(t)$ and $y_6(t)$ to be both equal to 0 for the 16 positions that appear in the previous parity check. Consider the case of register R_5 first. We are looking for states of R_5 at time t such that the corresponding outputs satisfy :

$$\begin{aligned}
y_5(t) &= 0,\\
y_5(t+T_1) &= 0,\\
y_5(t+T_2) &= 0,\\
y_5(t+T_3) &= 0,\\
y_5(t+T_4) &= 0,\\
y_5(t+T_1+T_2) &= 0,\\
y_5(t+T_1+T_3) &= 0,\\
y_5(t+T_1+T_4) &= 0,\\
y_5(t+T_2+T_3) &= 0,\\
y_5(t+T_2+T_4) &= 0,\\
y_5(t+T_3+T_4) &= 0,\\
y_5(t+T_1+T_2+T_3) &= 0,\\
y_5(t+T_1+T_2+T_4) &= 0,\\
y_5(t+T_1+T_3+T_4) &= 0,\\
y_5(t+T_2+T_3+T_4) &= 0,\\
y_5(t+T_1+T_2+T_3+T_4) &= 0.
\end{aligned}$$

If we enumerate the 2^{27} possible states of R_5 and clock the register T_{max} times, we can find all states that satisfy the above equations[3]. The expected number of solutions is

$$2^{27} \times 2^{-16} = 2^{11},$$

since there are 16 binary constraints to satisfy simultaneously. The complexity of this stage is about $2^{27} \times T_{max} = 2^{53.75}$. It is possible to do it more efficiently if we store the whole sequence of outputs from R_5, but this step will prove not to be the bottleneck of our attack. Similarly, we can find 2^{12} states of R_6 that satisfy the same 16 constraints. The corresponding time complexity is $2^{54.75}$. To summarize, we can enumerate

$$2^{12} \times 2^{11} = 2^{23}$$

favorable states for the registers R_5 and R_6. We store these 2^{23} states in an auxiliary table.

[3] We could envisage degenerated registers, for which these equations can never occur simultaneously. However, this is not the case in Achterbahn, and such degenerations would probably lead to other types of attacks.

In addition, for each favorable state, we clock R_5 and R_6 until we reach another favorable state. In the auxiliary table, we store the distance from each favorable state to the next one. This information will be useful in the next section. In average, we need 2^{32} clockings per favorable state, resulting in a time complexity of $2^{23} \times 2^{32} = 2^{55}$ steps.

4.4 Identification

We suppose that we are given a certain sequence of 2^{40} keystream bits. To simplify what follows, we start by computing the parity checks on the keystream bits,

$$\begin{aligned}
pc(t) &= z(t) + z(t+T_1) + z(t+T_2) + z(t+T_3) + z(t+T_4) \\
&\quad + z(t+T_1+T_2) + z(t+T_1+T_3) + z(t+T_1+T_4) \\
&\quad + z(t+T_2+T_3) + z(t+T_2+T_4) + z(t+T_3+T_4) \\
&\quad + z(t+T_1+T_2+T_3) + z(t+T_1+T_2+T_4) + z(t+T_1+T_3+T_4) \\
&\quad + z(t+T_2+T_3+T_4) + z(t+T_1+T_2+T_3+T_4),
\end{aligned}$$

for $t = 0 \ldots 2^{40} - T_{max}$.

It is very likely that R_5 and R_6 are in a favorable state, for at least one of the first 2^{32} positions in the sequence. We call t_0 such a position. Then we must have $pc(t_0) = 0$. This is only one bit of information, which is not sufficient to identify a favorable state.

Therefore, we enumerate all positions t_0 from 0 to 2^{32} and all the 2^{23} favorable states. Suppose we have $pc(t_0) = 0$ (otherwise we discard immediately the candidate). Then we use the auxiliary table to search for the next favorable state. Suppose the table says it will occur at the position $t_1 > t_0$. Then we jump to the position t_1 in the keystream sequence and check if $pc(t_1) = 0$. If it is not the case, we discard this candidate. Otherwise, we iterate the process.

Since we have 2^{40} keystream bits and the distance between two favorable states is about 2^{32}, we might be able to iterate up to $2^8 = 256$ times the process with success. This is sufficient to identify a favorable state, while a false alarm is very unlikely.

With our "early abort" strategy, we need to test only an average of 2 parity checks for each of the $2^{32} \times 2^{23} = 2^{55}$ candidates. So the time complexity of this phase is about 2^{56} steps.

4.5 Retrieving the key

We have identified the value of the state of R_5 and R_6 at a certain position t_0 in the output sequence. We would like to retrieve the key from this information, so a natural idea is to backtrack the updating of these registers. This is easy to do until we reach the initial state, since the update is invertible.

Next, we want to backtrack the initialization process of Achterbahn. During the extra clockings for diffusion and during the IV introduction, there is no difficulty to backtrack, since we can always predict the feedback bit. Unfortunately, we can no longer backtrack during the phase where the key was introduced.

Then, our idea is to perform a **meet-in-the-middle attack**. We split the key in two halves of 40 bits each. On the one hand, we guess the first 40 bits from the key and predict the state of R_5 and R_6 after the introduction of these 40 bits. On the other hand, we guess the last 40 bits from the key and backtrack the introduction of these bits from the known state of R_5 and R_6. We search for a match between the two lists of 2^{40} elements[4].

We should observe $2^{40} \times 2^{40} \times 2^{-55} \simeq 2^{25}$ matches since the lengthes of R_5 and R_6 sum up to 55 bits. Each of them provides a key candidate, which is easy to test by producing several keystream bits. To summarize, from one known state of R_5 and R_6, we can retrieve the secret key with time and memory complexity of 2^{40}.

4.6 Analysis

Both the precomputation and the identification phase of our attack have a time complexity of about 2^{56} steps. In addition, we need to store about 2^{40} (parity checks of) keystream bits and an auxiliary table of size 2^{23} after the precomputation phase.

The key recovery phase can be achieved using different trade-offs between time and memory. It is possible to do it with time and memory of 2^{40}. But a more reasonable trade-off could be with time 2^{50} and memory 2^{30}.

4.7 Cryptanalysis of full Achterbahn

If we want to attack the full Achterbahn, we must take into account the key-dependent linear combination used to compute the outputs of each register. This additional feature preserves the period of each registers, as well as the properties of the function F, so the observations on parity checks are unchanged. However, when looking for the favorable states of R_5 and R_6, we must guess in addition the $8 + 9 = 17$ key-dependent taps.

Depending on our guess on these key-dependent taps, we obtain a different set of favorable states. Therefore we must repeat 2^{17} times the second phase of our attack, and the whole complexity for attacking the full Achterbahn is about 2^{73} computation steps.

5 Another Cryptanalysis of Achterbahn

In this section, we propose another attack technique against Achterbahn, based on approximating its output function by a linear expression.

4 One bit is forced to 1 in each register to avoid the "all zero" state. The update is therefore not invertible, but we can easily guess the value of the erased bit, which has a negligible impact on the time complexity.

5.1 Linear approximations of the output function

Reconsider Achterbahn's output function given in Section 2,

$$\begin{aligned}z(t) &= F(y_1(t), y_2(t), y_3(t), y_4(t), y_5(t), y_6(t), y_7(t), y_8(t))\\ &= y_1(t) \oplus y_2(t) \oplus y_3(t) \oplus y_4(t) \oplus\\ &\quad y_5(t)y_7(t) \oplus y_6(t)y_7(t) \oplus y_6(t)y_8(t) \oplus y_5(t)y_6(t)y_7(t) \oplus y_6(t)y_7(t)y_8(t).\end{aligned}$$

We use the notation $l(t) = y_1(t) \oplus y_2(t) \oplus y_3(t) \oplus y_4(t)$ to refer to the linear part of F. it is easy to observe that F verifies the following linear approximations,

$$\begin{aligned}z(t) &= l(t) \oplus y_5(t) \ \text{ with probability } \ 10/16,\\ z(t) &= l(t) \oplus y_6(t) \ \text{ with probability } \ 12/16,\\ z(t) &= l(t) \oplus y_7(t) \ \text{ with probability } \ 12/16,\\ z(t) &= l(t) \oplus y_8(t) \ \text{ with probability } \ 10/16.\end{aligned}$$

In particular, we focus on the second approximation,

$$z(t) = l(t) \oplus y_6(t), \tag{1}$$

with probability $\frac{12}{16} = 0.75 = 0.5\ (1 + 0.5)$. Therefore the **bias** of this linear approximation is $\varepsilon = 0.5$.

5.2 Using the sparse parity checks

Similarly to Section 4.2, we can construct parity checks satisfied by the sequence of bits $l(t) \oplus y_6(t)$. Such a parity check will involve 32 keystream bits (instead of 16 like in Section 4.2) distant from at most

$$T_{max} = T_1 + T_2 + T_3 + T_4 + T_6 = 381681659 \simeq 2^{28.51}$$

positions. This parity check is not directly satisfied by the output sequence of Achterbahn since $l(t) \oplus y_6(t)$ is only an approximation of the output function. However we can sum up 32 times the linear approximation (1) over different values of t, which has the effect of multiplying the biases. Therefore, the parity check is satisfied by the sequence $z(t)$ with probability

$$0.5\ \left(1 + \varepsilon^{32}\right) = 0.5\ \left(1 + \frac{1}{2^{32}}\right).$$

Therefore if we consider a sequence of 2^{64} output bits and evaluate all the parity checks, we will detect this bias. This allows to distinguish Achterbahn's outputs from truly random sequences. In addition, this attack is not affected if we add key-dependent taps to each register, so its complexity is the same for the reduced and for the full Achterbahn.

5.3 Guessing one register

A natural extension of the previous distinguishing attack consists in guessing the initial content of register R_1 (there are 2^{23} candidates). Then, we can eliminate the term $y_1(t)$ in the previous linear approximation. Consequently, the weight of the parity check drops from 32 to 16, bringing the bias from 2^{-32} to 2^{-16}.

For the correct guess, we detect a bias by looking at 2^{32} keystream bits, while there is no bias for incorrect guesses. Once the correct guess has been identified, it is straightforward to repeat the process to target other registers. To summarize, this attack costs about 2^{55} computation steps and requires 2^{32} keystream bits. For the full Achterbahn, the number of guesses for R_1 is 2^{29} instead of 2^{23} increasing the complexity of the key recovery from 2^{55} to 2^{61}.

6 The case of Achterbahn-v2

6.1 Time-Memory Trade-Off

We reconsider the attack described in Section 4 in order to break the reduced version of Achterbahn-v2. Because of the linear complexity arguments, one can still construct the sparse parity checks satisfied by the linear part $l(t)$ (F and F' have both the same linear part). The criteria chosen by the designers is that it is no longer possible to cancel out the nonlinear part

$$nl(t) = z(t) \oplus l(t) \qquad (2)$$

by guessing 2 registers only, like for the "basic" Achterbahn. However, $nl(t)$ depends only on the initial state of the registers R_5, R_6, R_7 and R_8, which represents $27 + 28 + 29 + 31 = 115$ unknown bits. So we can apply the usual **time-memory-data trade-off** for stream ciphers :

– **Precomputation step** : Pick at random $2^{57.5}$ initial states for the registers R_5, R_6, R_7 and R_8 Then evaluate the parity check of Section 4.2 on the sequence of $nl(t)$ bits. We do this for 115 parity checks and we store the resulting vector of 115 bits in a table. Finally, the $2^{57.5}$ entries of this table are sorted according to the stored value.
– **Identification step** : Analyze a sequence of $2^{57.5}$ keystream bits and for each encountered position, evaluate the first 115 parity checks.

The parity check evaluated on the $z(t)$ bits is equal to the parity check evaluated on the $nl(t)$ bits, since it cancels out on the $l(t)$ bits (see relation (2)). Therefore, when we find a match, we learn the state of R_5, R_6, R_7 and R_8 during the encryption process. It turns out that a match is indeed expected here, due to the birthday paradox.

Besides, it is clear that the state of the 4 remaining registers, R_1, R_2, R_3 and R_4 as well as the secret key could be further retrieved, with an analysis similar to the one described in Section 4. We estimate the cost of this attack to $2^{57.5}$ in time and data complexity.

There is a technical detail to mention. Evaluating each parity check (in the precomputation step) requires to handle some $nl(t)$ bits which are located $T_{max} \simeq 2^{26.75}$ positions apart. To deal with this, we suggest to first compute the contribution of each separate register to the parity checks, for each possible state. This requires about $T_{max} \times 2^{31} \simeq 2^{57.75}$ steps for the longest register, *i.e.*, R_8. Then for each of the $2^{57.5}$ candidates, evaluating the 115 parity checks just requires several table look-ups and several XOR's. Arguably, the basic step in our attack costs about as much as testing one key in an exhaustive search.

This attack applies to the "reduced" Achterbahn-v2. Considering the case of the "full" Achterbahn-v2, we observe that each register has an "extra" entropy of $8, 9$ or 10 bits, due to the secret feed-forward. This sums up to 36 new unknowns. An exhaustive search over these unknowns is impossible, since it would bring the complexity above exhaustive search. Hence, we propose an alternative strategy in order to break the full Achterbahn-v2.

6.2 Breaking the full Achterbahn-v2

We propose a different extension of the attack of Section 4. The modification is that we target positions where $y_5(t) = y_6(t) = 1$ instead of 0. For these selected positions,

$$F'(y_1(t), \ldots, y_8(t)) = y_1(t) \oplus y_2(t) \oplus y_3(t) \oplus y_4(t) \oplus y_7(t) = \lambda(t)$$

So F' is reduced to a linear term $\lambda(t)$ with 5 terms (instead of 4 terms like $l(t)$) We view the bit $y_1(t) \oplus y_2(t)$ as the output of a single register with period $T_1 \cdot T_2$, so again we can write sparse parity checks of weight 16, involving terms located

$$T_{max} = T_1 \cdot T_2 + T_3 + T_4 + T_7 \simeq 2^{45}$$

positions apart. T_{max} is now much larger than previously, but as already pointed out in Section 4.3 the precomputation of "favorable" states for the register R_5 (or for R_6) can be done without clocking it T_{max} times (T_5 clockings are sufficient, taking into account its periodicity). Similarily, the increase of T_{max} does not change the time complexity of the identification step described in Section 4.4.

To summarize, the full Achterbahn-v2 can be attacked with the same complexity as the attack against the full Achterbahn described in Section 4.7, except that the data complexity is increased to $T_{max} = 2^{45}+2^{40} \simeq 2^{45}$ known keystream bits.

7 The case of Achterbahn-v3

Achterbahn-v3 has a new output function F'' which is not as sparse as its predecessors. However, F'' is approximable by a sparse linear function. Then the attack proposed in Section 5 can be applied. First, we observe that

$$z(t) = F''(y_1(t) \ldots y_8(t)) = y_1(t) \oplus y_2(t) \oplus y_3(t) \oplus y_4(t),$$

with probability $\frac{9}{16} = 0.5 \cdot (1 + \frac{1}{8})$. Then we apply the attack of Section 5 with the register-guessing trick. We guess the initial state of register R_1 (complexity 2^{22}), then we evaluate the parity check (of weight 8 since only 3 terms remain). The resulting bias is

$$\varepsilon = \left(\frac{1}{8}\right)^8 = 2^{-24},$$

so 2^{48} keystream bits are needed to detect the correct initial state of R_1. The time complexity of this attack is $2^{22} \times 2^{48} = 2^{70}$ for the reduced Achterbahn-v3. For the full Achterbahn-v3, there is an auxiliary factor of 2^6 to take into account for the secret feed-forward.

8 Conclusion

Type of Attack	*Technique*	*Target*	*Complexity*	*Data*
Key recovery (Sec. 4)	Linear Complexity	reduced Achterbahn	2^{56}	2^{40}
		full Achterbahn	2^{73}	2^{40}
Distinguisher (Sec. 5)	Linear Approx.	reduced Achterbahn	2^{64}	2^{64}
		full Achterbahn	2^{64}	2^{64}
Key recovery (Sec. 5)	Linear Approx.	reduced Achterbahn	2^{55}	2^{32}
		full Achterbahn	2^{61}	2^{32}
Key recovery (Sec. 6)	Time-Memory	reduced Achterbahn-v2	$2^{57.5}$	$2^{57.5}$
	Linear Complexity	full Achterbahn-v2	2^{73}	2^{45}
Key recovery (Sec. 7)	Linear Approx.	reduced Achterbahn-v3	2^{70}	2^{48}
		full Achterbahn-v3	2^{76}	2^{48}

Table 2. Summary of cryptanalysis results against Achterbahn

We proposed several attacks against Achterbahn and its modified versions. Table 2 summarizes all these cryptanalysis results. In spite of the nonlinear update, the fact that all registers are small and autonomous allows us to envisage several new attacks. Our idea is first to observe that a linear output function would give a low linear complexity and therefore an easily brekable cipher. Then we suggest to approximate the output function by a linear expression, and we build parity checks that the linearized version of Achterbahn should satisfy.

Following the publication of some preliminary results, the designers of Achterbahn suggested to modify the output filter. However, we pointed out that some attacks of the same nature are still possible. It is interesting to notice that our attacks are independent of the feedback of the nonlinear registers, so it illustrates some problems of the design itself, rather than an unfortunate instantiation.

References

1. Bluetooth. Bluetooth Specification, November 2003. available at http://www.bluetooth.org.
2. N. Courtois and W. Meier. Algebraic Attacks on Stream Ciphers with Linear Feedback. In E. Biham, editor, *Advances in Cryptology – Eurocrypt'03*, volume 2656 of *Lectures Notes in Computer Science*, pages 345–359. Springer, 2003.
3. eSTREAM - The ECRYPT Stream Cipher Project `http://www.ecrypt.eu.org/stream/`.
4. B. Gammel, R. Göttfert, and O. Kniffler. Improved Boolean Combining Functions for Achterbahn. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/072, 2005. http://www.ecrypt.eu.org/stream.
5. B. Gammel, R. Göttfert, and O. Kniffler. The Achterbahn Stream Cipher. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/002, 2005. http://www.ecrypt.eu.org/stream.
6. T. Johansson, W. Meier, and F. Muller. Cryptanalysis of Achterbahn. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/064, 2005. http://www.ecrypt.eu.org/stream.
7. J. Massey. Shift-Register Synthesis and BCH Decoding. *IEEE Transactions on Information Theory*, 15:122–127, 1969.
8. W. Meier and O. Staffelbach. Fast Correlations Attacks on Certain Stream Ciphers. In *Journal of Cryptology*, pages 159–176. Springer, 1989.
9. T. Siegenthaler. Correlation-immunity of Nonlinear Combining Functions for Cryptographic Applications. In *IEEE Transactions on Information Theory*, volume 30, pages 776–780, 1984.