# 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] DCSSI Crypto Lab 51, Boulevard de Latour-Maubourg
75700 Paris 07 SP France
Frederic.Muller@sgdn.pm.gouv.fr

**Abstract.** We present several attacks against Achterbahn, one of the new stream ciphers proposed to the eSTREAM competition. Our best attack breaks the reduced version of the cipher with complexity of $2^{56}$ steps and the full version with complexity of $2^{73}$ steps.
It highlights some problems in the design principle of Achterbahn, *i.e.* combining the outputs of several non-linear (but small) shift registers using a non-linear (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 [4]. 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 suggest that the design principle for Achterbahn, *i.e.* combining several small, non-linear shift registers by a very sparse combining function does not offer a satisfactory level of security.

## 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. The total size of the internal state is of 211 bits. At the $t$-th clock cycle, each register produces one output bit, denoted respectively by $y_1(t), \ldots, y_8(t)$. Then, the $t$-th output byte $z(t)$ of the stream cipher Achterbahn is produced by the following filtering function $F$ :

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

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 with 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.

### 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. In order to introduce this auxiliary input bit, it is simply XORed with the feedback bit during the register update. Before the encryption starts, several extra clockings are applied for diffusion purpose.

## 3 Weakness 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 originate from the linearity of the LFSR's. For instance, correlation attacks [6, 7] 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 be overcome. To improve the designs, it is often suggested to replace linear registers by non-linear registers. This idea is the bottomline of Achterbahn's design.

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

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

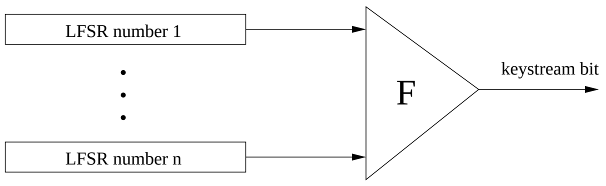

**Fig. 1.** Stream Cipher built by Combination of LFSR's

## 3.2 Linear Complexity of Achterbahn

If the linear registers of Figure 1 are replaced by non-linear 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 non-linear 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 given by

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

where $L_i$ denotes the linear complexity of each single register. This observation shows that **it would be insecure to combine the small non-linear registers using a linear function**. Indeed, in this case, the linear complexity $L$ of Achterbahn would be bounded by $8 \times 2^{31}$ since 31 is the length of the largest register.

For Achterbahn, $F$ is not linear, but its algebraic degree is 3. The original paper [4] does not contain an exact proof of the linear complexity of the 8 non-linear registers, but it is reasonable to assume that $L_i \simeq 2^{n_i}$ where $n_i$ denotes the length of register $R_i$. Even with this assumption, the linear complexity of Achterbahn's outputs is only :

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

If we apply the Berlekamp-Massey algorithm [5], 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 [4]). 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 non-linear 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 non-linear 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.

## 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 :

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

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 [4], 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

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

$llll(t)$ 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. 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.

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 the following, we start by computing the parity checks on the keystream bits :

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

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 $t_o$ such a positions, 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.

We should observe $2^{40} \times 2^{40} \times 2^{-55} \simeq 2^{25}$ matches since the length 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 $2^{30}$

## 5 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.

## 6 Another Cryptanalysis of Achterbahn

In this Section, we propose another attack technique against Achterbahn, based on approximating its output function by a linear expression. It is less efficient than the previous attack, and provides only a distinguisher. However it applies similarly for the full and the reduced Achterbahn, and reveals another weakness of the algorithm.

### 6.1 Linear approximations of the output function

Reconsider the 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 :

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

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 $\varepsilon = 0.5$

### 6.2 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 combine 32 times the linear approximation (1), 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' 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.

## 7 Conclusion

We proposed several attacks against Achterbahn. In spite of the non-linear update, the fact that all registers are small and autonomous seems to be a major problem. Our idea is first to observe that a linear output function would give a low linear complexity and therefore allow several attacks in spite of the non-linearity of the registers. 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.

Our best attack is based on guessing the state of two registers to simplify the linearization of the output function. It allows to recover the key for the reduced version of Achterbahn with complexity of about $2^{56}$ steps (while Achterbahn uses a key of 80 bits and a state of 211 bits). It also breaks the full Achterbahn with complexity of $2^{73}$ steps.

In addition, we propose a distinguishing attack based on linear approximations of Achterbahn. It requires to process about $2^{64}$ keystream bits and works identically for the reduced and the full Achterbahn.

It is interesting to notice that this attack is independent of the feedback of the non-linear registers, so it illustrates important flaws in 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. The Achterbahn Stream Cipher. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/002, 2005. http://www.ecrypt.eu.org/stream.
5. J. Massey. Shift-Register Synthesis and BCH Decoding. *IEEE Transactions on Information Theory*, 15:122–127, 1969.
6. W. Meier and O. Staffelbach. Fast Correlations Attacks on Certain Stream Ciphers. In *Journal of Cryptology*, pages 159–176. Springer, 1989.
7. T. Siegenthaler. Correlation-immunity of Nonlinear Combining Functions for Cryptographic Applications. In *IEEE Transactions on Information Theory*, volume 30, pages 776–780, 1984.