CRYPTANALYSIS OF A NONLINEAR FILTER-BASED STREAM CIPHER

TIM BEYNE AND MICHIEL VERBAUWHEDE

Abstract. It is shown that the stream cipher proposed by Carlet and Sarkar in ePrint report 2025/160 is insecure. More precisely, one bit of the key can be deduced from a few keystream bytes. This property extends to an efficient key-recovery attack. For example, for the proposal with 80 bit keys, a few kilobytes of keystream material are sufficient to recover half of the key.

Carlet and Sarkar [1] have recently proposed a stream cipher based on the nonlinear filter model. The filter function is based on the Maiorana-McFarland construction and the majority function. The authors propose several concrete instances with key sizes ranging from 80 to 256 bits. The parameters are chosen to ensure security against attacks that can use up to 2^{64} keystream bits per initialization vector and per key.

This note shows that this proposal is insecure if the attacker can insert a difference in the initialization vector, which is a standard assumption in the analysis of stream ciphers. The data- and time-complexity of the attack are practical. It exploits two weaknesses of the design: (1) the initialization is far too weak (2) although the filter function has good cryptanalytic properties, this does not carry over to the sum of two consecutive outputs of the filter function.

A single bit of the key can be recovered with high probability given two keystream bits for a handful of IV pairs. A similar approach recovers three other bits of the key. For a full key-recovery attack (with negligible time-complexity), a few kilobytes of keystream material are sufficient. An implementation for the parameter set with 80-bit keys can be found at the end of this note.

The attack combines a truncated differential over the initialization phase with a weakness of the sum of two filter functions for consecutive states. The latter aspect is analyzed in Section 1. The analysis of the initialization phase is contained in Section 2. Section 3 works out a full key recovery attack. This note assumes that the reader is already familiar with the specification of the cipher, which can be found in [1].

1. Analysis of the filter function for consecutive states

Let $\mathrm{maj}\colon \mathbf{F}_2^m \to \mathbf{F}_2$ denote the majority function on m bits. The filter function $f\colon \mathbf{F}_2^m \times \mathbf{F}_2^m \times \mathbf{F}_2 \to \mathbf{F}_2$ of the stream cipher is the following variant of the Maiorana-McFarland construction:

$$f(x, y, w) = x \cdot y + \mathrm{maj}(x) + w\,,$$

where $x \cdot y$ is the dot product between x and y. The values of x, y and w are taken from the current linear-feedback shift register state as illustrated in Figure 1. Here, l is the length of the state, and k is the key length. The filter function f has good linear and differential properties — the latter only if the difference on either x or y is nonzero. For this reason, our attack is based on sums of two consecutive keystream bits rather than on individual keystream bits. It can be interpreted as a differential-linear attack.

Tim Beyne is supported by the Research Foundation – Flanders (FWO) with reference № 1274724N. This work was partially supported by the Research Council KU Leuven, C16/18/004 through the C1 on New Block Cipher Structures. In addition, this work was supported by CyberSecurity Research Flanders with reference number VR20192203.

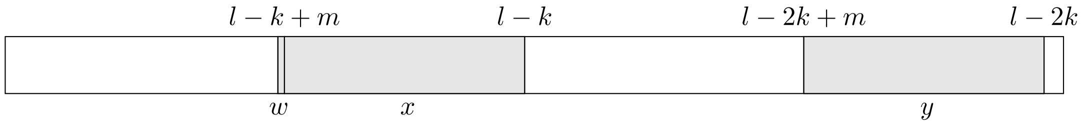


FIGURE 1. Layout of the variables x, y and w relative to the current state.

1.1. **Filter function for consecutive states.** The sum of two filter functions evaluated at consecutive states does not have good linear and differential properties. This can be exploited in a state recovery attack. However, to reduce the cost of the attack, we propose an additional truncated differential attack on the initialization phase in Section 2.

Let (x, y, w) and (x', y', w') be inputs to the filter function f for two consecutive states related by a linear-feedback shift register in Fibonacci mode. With this notation, the sum of the two output bits is equal to

$$f(x,y,w) + f(x',y',w') = x \cdot y + x' \cdot y' + \mathrm{maj}(x) + \mathrm{maj}(x') + w + w' .$$

The issue is that x and x', as well as y and y', are related by a shift. In particular, let s denote the state of the linear-feedback shift register, so that $x = (s_{l-k+m-1}, \ldots, s_{l-k})$ and $y = (s_{l-2k+m-1}, \ldots, s_{l-2k})$ as shown in Figure 1. Since $x' = (s_{l-k+m}, \ldots, s_{l-k-1})$ and $y' = (s_{l-2k+m}, \ldots, s_{l-2k-1})$, it follows that

$$x \cdot y + x' \cdot y' = s_{l-k}s_{l-2k} + s_{l-k+m}s_{l-2k+m} .$$

This is a quadratic function in only two variables. To deal with the majority function, let $g\colon \mathbf{F}_2^{m-1} \to \mathbf{F}_2$ be the Boolean function defined by

$$g(x) = \begin{cases} 1 & \text{if } \mathrm{wt}(x) = \lceil m/2 \rceil - 1\,, \\ 0 & \text{else}\,. \end{cases}$$

Importantly, this is a strongly unbalanced function. The majority function then satisfies

$$\mathrm{maj}(x) + \mathrm{maj}(x') = (s_{l-k} + s_{l-k+m})\, g(s_{l-k+m-1}, \ldots, s_{l-k+1}) .$$

Combining the results above with $w = s_{l-k+m}$ and $w' = s_{l-k+m+1}$ yields

$$\begin{aligned} f(x,y,w) + f(x',y',w') = {} & s_{l-k}s_{l-2k} + s_{l-k+m}(s_{l-2k+m} + 1) + s_{l-k+m+1} \\ & + (s_{l-k} + s_{l-k+m})\, g(s_{l-k+m-1}, \ldots, s_{l-k+1}) . \end{aligned}$$

This is a function in $m+4$ variables but, because g is strongly unbalanced, it is mostly determined by the five variables s_{l-2k}, s_{l-2k+m}, s_{l-k}, s_{l-k+m} and $s_{l-k+m+1}$. For brevity, denote this function by $q\colon \mathbf{F}_2^5 \to \mathbf{F}_2$, so that the sum of consecutive keystream bits is equal to

$$q(s_{l-2k}, s_{l-2k+m}, s_{l-k}, s_{l-k+m}, s_{l-k+m+1}) + (s_{l-k} + s_{l-k+m})\, g(s_{l-k+m-1}, \ldots, s_{l-k+1}) .$$

Let $h\colon \mathbf{F}_2^{m+4} \to \mathbf{F}_2$ denote the function mapping the vector with coordinates s_{l-2k}, s_{l-2k+m}, s_{l-k}, s_{l-k+m}, $s_{l-k+m+1}$ and $s_{l-k+m-1}, \ldots, s_{l-k+1}$ (in that order) to the above expression.

1.2. **Differential properties of q.** By direct computation, or by general properties of quadratic functions, one can see that the probability of every nontrivial differential for q is 1/2, except when the input difference is 00001. For the latter difference, the output difference is one with probability one.

1.3. **Differential properties of g.** Assume that m is odd. A combinatorial argument shows that if $\mathrm{wt}(a)$ is odd, then the probability of the differential $(a, 1)$ for g is equal to

$$\frac{1}{2^{m-2}}\binom{m-1}{\frac{m-1}{2}}.$$

If $\mathrm{wt}(a)$ is even, then the probability is

$$\frac{1}{2^{m-2}}\binom{m-1}{\frac{m-1}{2}} - \frac{1}{2^{m-2}}\binom{\mathrm{wt}(a)}{\frac{\mathrm{wt}(a)}{2}}\binom{m-1-\mathrm{wt}(a)}{\frac{m-1-\mathrm{wt}(a)}{2}}.$$

For large enough m, this probability is any case much smaller than 1/2. In particular,

$$\frac{1}{2^{m-2}}\binom{m-1}{\frac{m-1}{2}} \sim \frac{1}{\sqrt{\frac{\pi}{8}(m-1)}}.$$

For example, for $m = 37$ and $\mathrm{wt}(a) = 20$, the probability is approximately 19.5% (see Section 2).

1.4. **Differential properties of h.** For the function $h\colon \mathbf{F}_2^{m+4} \to \mathbf{F}_2$ that maps part of the linear-feedback shift register state to the sum of two consecutive keystream bits, one can compute the probabilities of differentials from those of q and g. For input differences of the form $00001 \parallel a$ with a in $\mathbf{F}_2^{m-1}$, the output difference is one with high probability. For other input differences, one expects that output differences zero and one occur with equal probability.

Let p be the probability that the output difference of g is equal to one. There are two differential characteristics, and no nontrivial quasidifferential trails [2]. Hence, the probability that the output difference of h is one equals

$$1 - p + \frac{p}{2} = 1 - \frac{p}{2}.$$

Explicitly, if $\mathrm{wt}(a)$ is odd, then

$$1 - \frac{1}{2^{m-1}}\binom{m-1}{\frac{m-1}{2}}.$$

If $\mathrm{wt}(a)$ is even, then

$$1 - \frac{1}{2^{m-1}}\binom{m-1}{\frac{m-1}{2}} + \frac{1}{2^{m-1}}\binom{\mathrm{wt}(a)}{\frac{\mathrm{wt}(a)}{2}}\binom{m-1-\mathrm{wt}(a)}{\frac{m-1-\mathrm{wt}(a)}{2}}.$$

In both cases, if m is large enough, then this probability is close to one.

2. Analysis of the initialization phase

The initialization phase is based on a nonlinear-feedback shift register and consists of $2k$ rounds. The key and initialization vector are placed in the state as indicated in Figure 2. The feedback function is the sum of the feedback of the linear-feedback shift register, and the output of the filter function. The analysis below shows that probability-one truncated differentials can be constructed for the entire initialization phase. These truncated differentials result in a constant (but key dependent) difference on some of the state bits. If the difference is of the form discussed in Section 1.4, then this can be detected using a few samples.

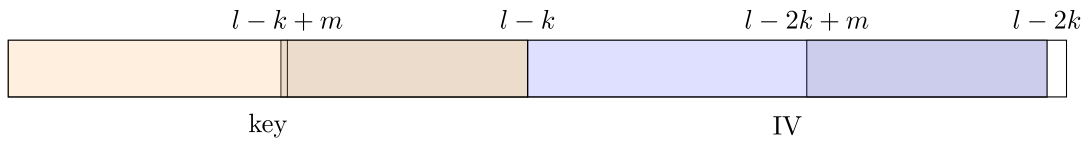


FIGURE 2. Initial state, with the first $l - 2k$ bits equal to a public constant.

The following analysis uses the parameters $l = 163$, $k = 80$ and $m = 37$ with feedback polynomial $x^{163} + x^7 + x^6 + x^3 + 1$ as an example. The attack also affects the other parameters from [1, Tables 1 and 2]. To recover the highest key bit, a difference is introduced in the highest bit of the initialization vector, as illustrated in Figure 3. Recovery of other key bits is discussed in Section 3.

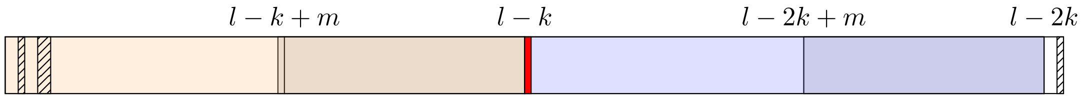


FIGURE 3. Input difference in the highest bit (red) of the initialization vector. The hatched bits indicate the tap positions of the linear-feedback shift register. Dark areas correspond to the inputs of the nonlinear feedback function.

2.1. **First $l-k$ initialization rounds.** In the first $k-m$ rounds, the difference simply moves to the right because the feedback function is inactive. In round $k-m+1$, a difference is introduced at the input of the filter function — specifically, in the part denoted by y in Figure 1. Since $f(x, y+a, w) = f(x, y, w) + a \cdot x$, the feedback function introduces a difference K_{k-1} equal to the highest key bit. In the next round, the same difference is once again introduced because the key shifts along with the difference on the initialization vector. In particular, if $K_{k-1} = 0$, then *no difference is introduced* for an additional $m + l - 2k$ rounds.

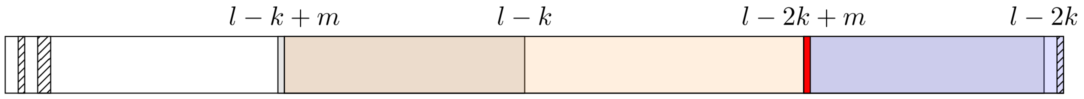


FIGURE 4. State difference after $k - m$ initialization rounds.

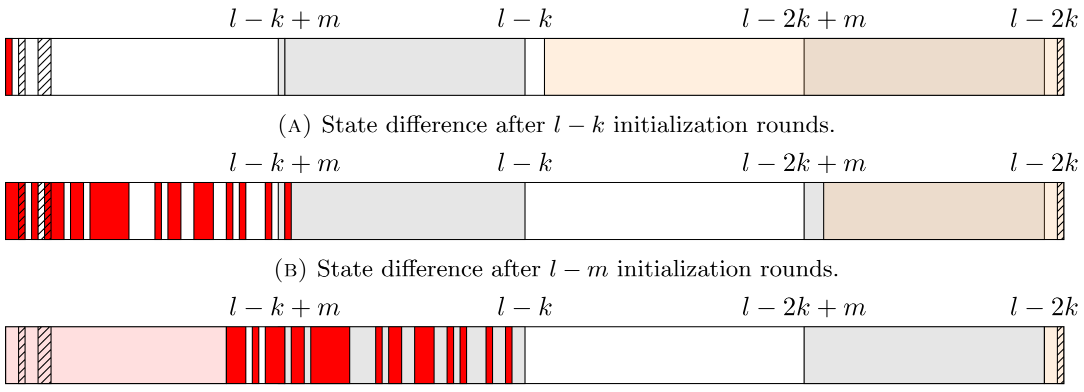
2.2. **Final $3k - l$ initialization rounds.** After $l - k$ rounds, the nonzero difference bit reaches the highest bit of the state as shown in Figure 5a. In the next $k - m$ rounds, the feedback function introduces differences in a deterministic manner. The result is shown in Figure 5b. For the remaining $m + 2k - l$ rounds, the differences that are introduced will be truncated. The resulting difference, with truncated differences lightly colored in red, is shown in Figure 5c.

2.3. **Recovering the highest bit of the key.** After the $2k$ initialization rounds, the production of keystream bits begins. To use the high-probability differential from Section 1.4, the input difference of the function $h\colon \mathbf{F}_2^{m+4} \to \mathbf{F}_2$ that maps part of the state to the sum of two consecutive keystream bits should be of the form $00001 \parallel a$ for a nonzero difference a.

For the concrete parameters above, this happens after three linear-feedback shift register clocks. The difference a then satisfies $\mathrm{wt}(a) = 20$. Hence, the difference on the sum of the third and fourth keystream bits is equal to one with probability

$$1 - \frac{1}{2^{36}}\binom{36}{18} + \frac{1}{2^{36}}\binom{20}{10}\binom{16}{8} \approx 0.903\,.$$

This holds whenever $K_{79} = 0$. Otherwise, the probability is close to $1/2$ (as verified by additional analysis, or experimentally). Distinguishing between these cases with high success probability and low false-positive probability requires less than 80 keystream bits — e.g. four keystream bits for ten IV pairs.



(A) State difference after $l-k$ initialization rounds.

(B) State difference after $l-m$ initialization rounds.

(C) State difference after $2k$ initialization rounds.

FIGURE 5. State differences during the final $3k-l$ initialization rounds.

3. Key recovery

The attack from Section 2.3 can be extended to a full key-recovery attack. This section works this out for the parameters $l = 163$, $k = 80$ and $m = 37$. In this case, 40 bits of the key are recovered with negligible computational cost.

3.1. **Recovering four key bits.**

The analysis from Section 2 allows recovering the highest bit of the key. Additional key bits can be recovered using the same approach, but by inserting a difference in lower bits of the initialization vector. The following table gives an overview of the number of clocks required and the corresponding differential probability.

Difference in	Condition	Clocks	Probability
IV_{79}	$K_{79} = 0$	3	0.903
IV_{78}	$K_{78} = 0$	2	0.903
IV_{77}	$K_{77} = 0$	1	0.903
IV_{76}	$K_{76} = 0$	0	0.903

Additional key bits can be recovered, but this requires some additional analysis because the third and fourth bit of the input difference of h are not uniquely determined when more than six clocks are required. Another approach is to inject a difference in two bits of the initialization vector.

3.2. **Recovering additional bits of the key.**

The following method can be used to recover 36 additional key bits. It assumes that at least one of the bits K_{79}, K_{78} and K_{77} is zero. There are ways to get around this restriction with little additional cost, but the discussion below focuses on this case for simplicity.

If $K_{80-i} = 0$ for i in $\{1, 2, 3\}$, then one can detect a difference on the sum of consecutive keystream bits after $12 - i$ additional clocks to recover 36 key bits. This requires introducing a difference in IV_{80-i}. The main observation is that the input difference of h depends on part of the IV. Specifically, it is of the following form:

$$000 \parallel \ell_i + \varepsilon_1 \parallel \ell_i + \varepsilon_2 + 1 \parallel a\,.$$

If the bits $\mathrm{IV}_{3-i}, \ldots, \mathrm{IV}_{m+2-i}$ are fixed, then ℓ_i is an affine function of the key up to addition by the majority function of part of the key:

$$\begin{aligned}\ell_i &= c_{3-i} + K_{77-i} + K_{76-i} + f(K_{m+2-i,\ldots,3-i}, \mathrm{IV}_{m+2-i,\ldots,3-i}, K_{m+3-i}) \\ &= c_{3-i} + K_{77-i} + K_{76-i} + K_{m+3-i} + \mathrm{maj}(K_{m+2-i,\ldots,3-i}) + \sum_{j=3-i}^{m+2-i} \mathrm{IV}_j K_j .\end{aligned}$$

The weight of a is 22 if $\ell_i = \varepsilon_0$ and 23 otherwise. Here, the variables ε_j with j in $\{0, 1, 2\}$ are unbalanced functions of the state after $127 - j$ clocks. Specifically,

$$\varepsilon_j = \mathrm{maj}(s_{l-k+m+j}, \ldots, s_{l-k-1+j}) + \mathrm{maj}((s_{l-k+m+j}, \ldots, s_{l-k-1+j}) + \delta_{j+1}) \,,$$

where δ_{j+1} is the $j + 1^{\text{st}}$ standard basis vector. By determining the value of ℓ_i for 37 linearly independent choices of $(\mathrm{IV}_{2-i}, \ldots, \mathrm{IV}_{m+2-i})$, one can recover 36 bits of the key by solving a system of linear equations — the variable $\mathrm{maj}(K_{m+2-i,\ldots,2-i})$ can be eliminated.

The data-complexity is determined by the probability that the difference between the sum of the two consecutive keystream bits is equal to one. Since ε_0, ε_1 and ε_2 are biased towards zero, the probability is significantly higher for $\ell_i = 0$ than for $\ell_i = 1$. As before, if the input difference is $00001 \parallel a$, then the probability that the output difference of h is one is equal to 0.903 for $\mathrm{wt}(a) = 22$ and 0.868 for $\mathrm{wt}(a) = 23$ respectively. For other input differences, the probability is close to 1/2. A rough estimate of the probabilities can be derived from the distribution of $(\boldsymbol{\varepsilon}_0, \boldsymbol{\varepsilon}_1, \boldsymbol{\varepsilon}_2)$ for a uniform random state, assuming these values are independent of the input of h. This yields probabilities 0.82 if $\ell_i = 0$ and 0.53 if $\ell_i = 1$.

Table 1. Distribution of $(\boldsymbol{\varepsilon}_0, \boldsymbol{\varepsilon}_1, \boldsymbol{\varepsilon}_2)$ if the state is uniform random.

ε_0	ε_1	ε_2	Probability
0	0	0	0.753
0	0	1	0.049
0	1	0	0.033
0	1	1	0.033
1	0	0	0.049
1	0	1	0.017
1	1	0	0.033
1	1	1	0.033

3.3. Cost estimate.

Recovering the 36 additional key bits requires $37N$ pairs of initialization vectors, and $13 - i$ keystream bits for each. Here, N is the number of pairs necessary to distinguish between events with probabilities 0.82 and 0.53. There is a trade-off between N and the time required to solve a system of linear equations with some errors. A few kilobytes of keystream material are sufficient.

References

[1] Claude Carlet and Palash Sarkar. The nonlinear filter model of stream cipher redivivus. Cryptology ePrint Archive, Paper 2025/160, 2025.

[2] Tim Beyne and Vincent Rijmen. Differential cryptanalysis in the fixed-key model. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part III*, volume 13509 of *Lecture Notes in Computer Science*, pages 687–716. Springer, 2022.

Appendix A. Implementation for $k = 80$

```
from functools import partial

# This code only works for the following parameters:
(L, k, m) = (163, 80, 37)
# Any constant
c = [1, 1, 1]
# Primitive polynomial for LFSR
x = polygen(GF(2))
f = x^163 + x^7 + x^6 + x^3 + 1

bound = ceil(m / 2)
M = companion_matrix(f.reverse(), format = 'bottom')

def lfsr(x):
    return M*x

def filter(state):
    w = state[L - k + m]
    x = state[L - k : L - k + m]
    y = state[L - 2*k : L - 2*k + m]
    maj = 1 if x.list().count(1) >= bound else 0
    return x * y + maj + w + 1

def full_init(key, iv, c):
    state = vector(GF(2), c + iv + key)
    for _ in range(2*k):
        f = filter(state)
        state = lfsr(state)
        state[-1] += f
    return state

def get_data(key, iv, l):
    state = full_init(key, iv, c)
    res = [filter(state)]
    for _ in range(l - 1):
        state = lfsr(state)
        res.append(filter(state))
    return res

def get_keybit(f, i):
    assert(i >= 76 and i < 80)
    t = i - 76
    count = 0
    for _ in range(32):
        iv = [randrange(2) for _ in range(k)]
        data1 = f(iv, t+2)
        iv[i] ^^= 1
        data2 = f(iv, t+2)
        if data1[t] + data1[t+1] + data2[t] + data2[t+1]:
            count += 1
    return 0 if count >= 24 else 1

def get_more_keybits(f, i):
    t = 80 - i
    assert(t >= 1 and t < 4)
    res = []
    # Lower data possible if we try to correct errors (LPN/decoding problem),
    # around 308 IV pairs
    for j in range(m):
        count = 0
        for _ in range(93):
            iv = [randrange(2) for _ in range(k)]
            for l in range(m):
                iv[3-t+l] = 1 if l == j else 0
            data1 = f(iv, 11-t+2)
            iv[i] ^^= 1
            data2 = f(iv, 11-t+2)
            if data1[-1] + data1[-2] + data2[-1] + data2[-2]:
                count += 1

        key_eq = vector(GF(2), 80, {77 - t : 1, 76 - t : 1, m + 3 - t : 1, 3 - t + j : 1})
        res.append((key_eq, GF(2)(c[2]) if count >= 62 else GF(2)(c[2])+1))

    final_res = []
    for i in range(36):
        final_res.append((res[i][0] + res[-1][0], res[i][1] + res[-1][1]))
    return final_res

def key_recovery(bit):
    key = [randrange(2) for _ in range(k)]
    key[bit] = 0
    for i in range(76, k):
        print(i, key[i] == get_keybit(partial(get_data, key), i))
    res = get_more_keybits(partial(get_data, key), bit)
    keyv = vector(GF(2), key)
    print(all(eq*keyv == c for eq, c in res))

key_recovery(79)
key_recovery(78)
key_recovery(77)
```

COSIC, KU Leuven
Email address: name.lastname@esat.kuleuven.be