Attacking the filter generator by finding zero inputs of the filtering function

Frédéric Didier *

Projet CODES, INRIA Rocquencourt, Domaine de Voluceau,
78153 Le Chesnay cedex
frederic.didier@inria.fr

Abstract. The filter generator is an important building block in many stream ciphers. We present here an attack that recovers the initial state of the hidden LFSR by detecting the positions where the inputs of the filtering function are equal to zero. This attack requires the precomputation of low weight multiples of the LFSR generating polynomial. By a careful analysis, we show that the attack complexity is among the best known and work for almost all cryptographic filtering functions.

Keywords: Stream cipher, filter generator, Boolean functions, low weight multiples, autocorrelation.

1 Introduction

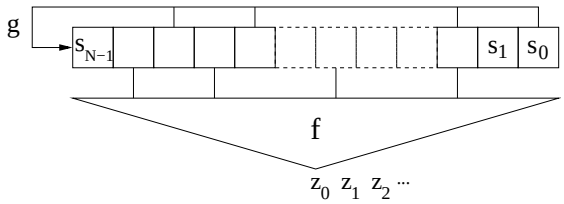

Fig. 1. LFSR filter generator.

The filter generator uses a linear feedback shift register (LFSR) of length N and characteristic polynomial $g(X)$ that generates a binary sequence $(s_t)_{t\geq 0}$ of period 2^N-1. As we can see in Figure 1 this sequence is filtered using a n-variable balanced Boolean function f (from $\mathbf{F}_2^n$ into $\mathbf{F}_2$) to produce the keystream $(z_t)_{t\geq 0}$. The inputs of this function are taken as some bits in the LFSR internal state. We will write $\mathbf{x}_t$ for the n-bit vector corresponding to the inputs of f at time t. Notice that we will always write such elements of $\mathbf{F}_2^n$ in bold. Our goal here is to

* This work is partially funded by CELAR/DGA.

find the key (that is the LFSR initial state $s_0, \ldots, s_{N-1}$) knowing the keystream sequence $(z_t)_{t\geq 0}$ and all the constituents of the filter generator.

The filter generator is one of the simplest stream cipher and it is really interesting to understand what kind of attacks we may perform on it. There is of course a huge literature on the subject and quite a few approaches. Some of the most important ones fall into the category of fast correlation attacks. They were introduced by Meier and Staffelbach [MS88] as an improvement to correlation attack introduced by Siegenthaler in [Sie85]. Since then, many different versions have been proposed (see for instance [CT00], [MFI01],[CF02],[JJ00] and [JJ02]). The other main class is given by the algebraic [CM03] and fast algebraic attacks [Cou03] which can be really efficient if the filtering function is of low algebraic immunity. Recently, Rønjom and Helleseth have proposed a new variant [RH07] which is closely related to the Berlekamp Massey attack. There is also some ideas in [MFI05] and [MFI06] that apply to the two previous categories of attacks. When the inputs positions of f (also known as tapping positions) are not well chosen, one can apply inversion attack or conditional correlation attack (see [Gol96],[GCD00] and [LCPP96]). Finally, there is the very general class of time-memory-data tradeoff attack (see [BS00]) which is often the most efficient if the generator is well designed.

In this paper, we will present a new attack related to vectorial versions of fast correlation attacks (see [LZGB03], [EJ04], [GH05]). This attack has an interesting complexity and appears to be difficult to avoid. The idea is to use the low degree multiples to distinguish the positions corresponding to zero inputs of the filtering function f. Most of the probabilistic analysis is derived from the work of Sabine Leveiller during her PhD [Lev04b] (it is in French but some of it is published in [Lev04a] and [LZGB03]). However, we push it a little further and show that the positions corresponding to zero inputs of f are actually almost always detectable.

The paper is organized as follows. We begin by explaining the attack principle in the first section. Then, in Section 2, we compute the bias at the heart of the attack, this is our main contribution. This also allows us to derive the actual attack complexity in Section 3. We give in Section 4 the time complexity of such an attack on some example filter generators. We finally conclude in the last section.

2 Attack principle

Our attack uses like many correlation attacks the small weight multiples of the polynomial $g(X)$ generating the LFSR. Each of these multiples induces a linear relation between some points where f is applied to produce the keystream. Namely, for a multiple $p(X) = 1 + \sum_{i=1}^{w} X^{p_i}$ of weight $w+1$ we have

$$\mathbf{x}_t + \mathbf{x}_{t+p_1} + \cdots + \mathbf{x}_{t+p_w} = \mathbf{0} \quad \forall t \geq 0 \tag{1}$$

In all this paper, we will assume that for a given multiple and a point $\mathbf{x}_t$, the others point $\mathbf{x}_{t+p_1}$ up to $\mathbf{x}_{t+p_w}$ can take with the same probability any value

satisfying (1). This is justified by the good properties of an LFSR sequence and appears to be a good working hypothesis since we will see that the experimental results are very close to the predicted ones. With this assumption, we define

$$P_{\mathbf{x}} \stackrel{\text{def}}{=} \Pr\left(f(\mathbf{x}_1) + \cdots + f(\mathbf{x}_w) = 0 \quad | \quad \sum_{i=1}^{w} \mathbf{x}_i = \mathbf{x} \right). \tag{2}$$

We did not include the p_i in this expression because they have no real influence in this model. Actually in our model $P_{\mathbf{x}}$ is exactly the probability that for a given multiple and a time t, $z_{t+p_1} + \cdots + z_{t+p_w}$ is equal to zero knowing that $\mathbf{x}_t = \mathbf{x}$. The crux of our attack is based on these probabilities. They can be expressed nicely as we will see in the next section and for an even w they satisfy two interesting properties:

- $P_{\mathbf{0}}$ is always greater than $1/2$ and is greater than or equal to the other $P_{\mathbf{x}}$'s.
- If the function f has a good autocorrelation property then there is always a gap between $P_{\mathbf{0}}$ and the other $P_{\mathbf{x}}$'s.

At this point, one could guess what we are going to do. Using many multiples of g, we will be able to have a good approximation of the probability $P_{\mathbf{x}_t}$ associated to a position t. Now, if the gap between $P_{\mathbf{0}}$ and the others $P_{\mathbf{x}}$ is large enough (depending of the number of multiples we used) we will then be able to detect which time positions are associated with an $\mathbf{x}_t$ equal to $\mathbf{0}$.

Each $\mathbf{x}_t$ equal to $\mathbf{0}$ actually tells us that the n bits of the sequence $(s_t)_{t\geq 0}$ involved in this $\mathbf{x}_t$ are equal to 0. By substituting their linear expression in terms of $s_0, \ldots, s_{N-1}$ we then obtain n linear equations involving the key bits. In the end, merging the equations from all the zero $\mathbf{x}_t$'s, we get a linear system of rank at most $N-1$ since both the all zero state and the actual LFSR initial state are solutions. We thus hope that given $\lceil N/n \rceil$ such zero $\mathbf{x}_t$, we should get a rank $N-1$ linear system where the only non trivial solution is the LFSR initial state.

The attack algorithm is summarized here with two parameters D and L that will be discussed later:

1. Compute all the weight $2p+1$ multiples of $g(X)$ up to degree D. This can be done offline and once for all.
2. Approximate $P_{\mathbf{x}_t}$ for the L first bits of the keystream. In order to do that, for a given position each multiple corresponds to a parity check, and we just have to count how many are satisfied by the keystream bits. Remark that among the L bits, only the ones for which $z_t = f(\mathbf{0})$ have to be considered.
3. Assume that the $\lceil N/n \rceil$ bits with the higher approximated $P_{\mathbf{x}_t}$ correspond to positions where $\mathbf{x}_t = 0$.
4. Solve the linear system induced by the knowledge of $\mathbf{x}_t$ at these positions and retrieve the initial state.

A detailed complexity analysis will be carried in the following sections but we give here a preliminary one. The best complexity for the first step (precomputation) is given by the algorithm of [CJM02] and is in D^p time and $D^{p/2}$

memory. The complexity for Step 2 is in L times the number of multiples and requires the knowledge of $D+L$ keystream bits. The last two steps are negligible in the overall complexity. Remark however that last step may deal with some erroneous answers at Step 3 by trying more than one linear system induced by the positions with a high $P_{\mathbf{x}_t}$.

3 Bias computation

We will give here a simple expression for the probability

$$P_{\mathbf{x}} \stackrel{\text{def}}{=} \Pr\left(f(\mathbf{x}_1)+\cdots+f(\mathbf{x}_w)=0 \quad | \quad \sum_{i=1}^{w}\mathbf{x}_i=\mathbf{x}\right) \tag{3}$$

corresponding to an equation of weight $w+1$. We will then use it to compute the gap between $P_{\mathbf{0}}$ and the other $P_{\mathbf{x}}$ in the case of an even w. In order to do that, let us introduce

$$d_w(\mathbf{x}) \stackrel{\text{def}}{=} \sum_{\mathbf{x}_1,\ldots,\mathbf{x}_{w-1}\in\mathbf{F}_2^n} (-1)^{f(\mathbf{x}_1)+\cdots+f(\mathbf{x}_{w-1})+f(\mathbf{x}+\mathbf{x}_1+\cdots+\mathbf{x}_{w-1})} \tag{4}$$

where $\mathbf{x}+\mathbf{x}_1\cdots+\mathbf{x}_{w-1}$ corresponds to $\mathbf{x}_w$ in (3) since the sum of $\mathbf{x}_1$ up to $\mathbf{x}_w$ must be equal to $\mathbf{x}$. By definition, d_1 is the sign function of f

$$d_1(\mathbf{x}) = (-1)^{f(\mathbf{x})} \tag{5}$$

and the d_w's are directly related to the probability $P_{\mathbf{x}}$ by

$$P_{\mathbf{x}} = \frac{1}{2}\left(1+\frac{1}{2^{(w-1)n}}d_w(\mathbf{x})\right). \tag{6}$$

Moreover, it is easy to show the following recursive relation

$$d_w(\mathbf{x}) = \sum_{\mathbf{y}\in\mathbf{F}_2^n}(-1)^{f(\mathbf{x}+\mathbf{y})}d_{w-1}(\mathbf{y}) = d_1 * d_{w-1}(\mathbf{x}) \tag{7}$$

where $*$ is the convolution product. Using the properties of the Walsh transform we obtain that $\widehat{d_w}(u) = \widehat{d_1}(u)^w$ where

$$\widehat{d_1}(\mathbf{u}) = \sum_{\mathbf{x}\in\mathbf{F}_2^n} d_1(\mathbf{x})(-1)^{\mathbf{u}.\mathbf{x}} \tag{8}$$

And by using the inverse Walsh transform we finally obtain

$$d_w(\mathbf{x}) = \frac{1}{2^n}\sum_{\mathbf{u}\in\mathbf{F}_2^n}(-1)^{\mathbf{u}.\mathbf{x}}\widehat{d_1}(\mathbf{u})^w. \tag{9}$$

That is

$$P_{\mathbf{x}} = \frac{1}{2}\left[1+\sum_{\mathbf{u}\in\mathbf{F}_2^n}(-1)^{\mathbf{u}.\mathbf{x}}\left(\frac{\widehat{d_1}(\mathbf{u})}{2^n}\right)^w\right]. \tag{10}$$

This has already be observed by Sabine Leveiller in her PhD thesis [Lev04b], we have just given here another proof of this statement. For reference, one may look at [LZGB03] since her PhD is in French only.

We will show now that the probability $P_{\mathbf{0}}$ can be distinguished quite well from the others when w is even. For that it is natural to look at the minimum difference between $P_{\mathbf{0}}$ and $P_{\mathbf{x}}$, that is to compute $\min_{\mathbf{x}\neq\mathbf{0}}(P_{\mathbf{0}} - P_{\mathbf{x}})$.

Let us begin by defining Δ to be the minimum of $P_{\mathbf{0}} - P_{\mathbf{x}}$ when $w = 2$. Using (6) we have

$$\Delta \stackrel{\text{def}}{=} \frac{1}{2}\left[\frac{d_2(\mathbf{0})}{2^n} - \max_{\mathbf{x}\neq\mathbf{0}}\left(\frac{d_2(\mathbf{x})}{2^n}\right)\right] = \frac{1}{2}\left[1 - \max_{\mathbf{x}\neq\mathbf{0}}\left(\frac{d_2(\mathbf{x})}{2^n}\right)\right] \tag{11}$$

since $d_2(\mathbf{0})$ is equal to 2^n by Parseval's equality. Usually, if f has a good autocorrelation then this Δ is very close to $1/2$. To see that, looking at the formula (4) we have

$$d_2(\mathbf{x}) = \sum_{\mathbf{u}} (-1)^{f(\mathbf{u})+f(\mathbf{u}+\mathbf{x})} \tag{12}$$

which is nothing more than an autocorrelation coefficient and should be close to 0. Remark that if this is not the case we are confident that we can distinguish quite well other values of $P_{\mathbf{x}}$ from the others.

In the more general case $w = 2p$, we can write the difference between $P_{\mathbf{0}}$ and $P_{\mathbf{x}}$ as

$$\min_{\mathbf{x}\neq\mathbf{0}}(P_{\mathbf{0}} - P_{\mathbf{x}}) = \frac{1}{2}\min_{\mathbf{x}\neq\mathbf{0}}\left[\sum_{\mathbf{u}\in\mathbf{F}_2^n}\left(\frac{\widehat{d_1}(\mathbf{u})}{2^n}\right)^{2p} - \sum_{\mathbf{u}\in\mathbf{F}_2^n}(-1)^{\mathbf{u}.\mathbf{x}}\left(\frac{\widehat{d_1}(\mathbf{u})}{2^n}\right)^{2p}\right] \tag{13}$$

that is,

$$\min_{\mathbf{x}\neq\mathbf{0}}(P_{\mathbf{0}} - P_{\mathbf{x}}) = \min_{\mathbf{x}\neq\mathbf{0}}\sum_{\mathbf{u},\mathbf{u}.\mathbf{x}=1}\left(\frac{\widehat{d_1}(\mathbf{u})}{2^n}\right)^{2p}. \tag{14}$$

Notice that this difference is always greater than or equal to 0 which means that $P_{\mathbf{0}}$ is always greater than or equal to the other probabilities. In the case $p = 1$ this is nothing more than Δ and using the Hölder inequality (see Appendix A) we can see that the worst case for the others p is when all the $\widehat{d_1}(\mathbf{u})$ are equal. Since there is 2^{n-1} terms in the sum, the worst case is when for $p = 1$ each term in the sum is equal to $\Delta/2^{n-1}$. We thus get this lower bound

$$\min_{\mathbf{x}\neq\mathbf{0}}(P_{\mathbf{0}} - P_{\mathbf{x}}) \geq 2^{n-1}\left(\frac{\Delta}{2^{n-1}}\right)^p \tag{15}$$

which corresponds to the bias we will need to detect.

4 Complexity analysis

Let us look at the complexity of attacking the filter generator when we use multiples of weight $2p + 1$. We will suppose that the function has a good auto-

correlation property, meaning that the bias to detect is around

$$\text{bias} \simeq \frac{1}{2^{1+n(p-1)}}. \tag{16}$$

To detect it, we will thus need as many equations as the square of the bias inverse. Looking for multiples of weight $2p+1$ up to degree D of the LFSR generator polynomial, we know that we will find around

$$\text{degree at most D multiples number} \simeq \binom{D}{2p} \frac{1}{2^N} \simeq \frac{D^{2p}}{(2p)!2^N} \tag{17}$$

of them. This result is well known and is derived as follows. We have $\binom{D}{2p}$ polynomials of weight $2p+1$ and degree at most D. For each of them, we may assume that the rest of the Euclidean division by $g(X)$ is equally distributed among the 2^N possible values. Hence, the formula (17) just express that we get a multiple (a rest equal to 0) one time over 2^N.

Putting the equations (16) and (17) together, to be able to detect our bias we need to choose a degree D such that

$$\frac{D^{2p}}{(2p)!2^N} = 2^{2+2n(p-1)}. \tag{18}$$

That means we will have to compute the weight $2p+1$ multiples up to a degree D where

$$\log_2 D = \frac{N}{2p} + n\left(1 - \frac{1}{p}\right) + \frac{1}{p}. \tag{19}$$

We neglect the factorial term $(2p)!$ here since in practice p is 2 or 3. Using the algorithm of [CJM02], the complexity to compute them is in D^p time and $D^{p/2}$ memory. Remark that the algorithm is completely parallelizable over many computers. We finally get for the offline part of the attack

$$\log_2(\text{offline time}) = N/2 + (p-1)n + 1 \tag{20}$$

$$\log_2(\text{offline memory}) = N/4 + (p-1)n/2 + 1/2. \tag{21}$$

For the online phase, we will need to identify around $\lceil N/n \rceil$ bits corresponding to an $\mathbf{x}$ equal to $\mathbf{0}$. We will thus need to approximate the $P_{\mathbf{x}_t}$ for L bits in average where

$$L = \left\lceil \frac{N}{n} \right\rceil 2^n. \tag{22}$$

This comes from the fact that an $\mathbf{x}$ equal to $\mathbf{0}$ appears in average one time each 2^n keystream bits. We can actually gain a factor 2 because we can skip the positions for which $f(\mathbf{x}_t) \neq f(\mathbf{0})$. For each of these L bits, we will have to compute as many parity checks as the number of multiples. The online phase complexity is then given by

$$\log_2(\text{online time}/L) = 2 + 2n(p-1) \tag{23}$$

which is in practice really efficient. For the memory we just need to access the stored multiples and a length of keystream equal to $D + L$, that is basically

$$\log_2(\text{keystream length}) = \frac{N}{2p} + n\left(1 - \frac{1}{p}\right) + \frac{1}{p}. \tag{24}$$

Remark that the overall complexity is quite good. Let us compare it with the time-memory-data tradeoff described in [BS00]. This tradeoff is such that $TM^2K = 2^{2N}$ where T is the online time, M the online memory and K the length of the keystream needed. A good choice is to take $M = K = 2^{N/3}$ which gives an online time of $2^{2N/3}$ and the same precomputation time. For our attack with weight 5 multiples and an n around $N/8$ (which is typical), we have the following: an online time complexity in $2^{N/4}$ and memory in $2^{5N/16}$ for a keystream length of $2^{5N/16}$ bits. This is better than the time-memory-data tradeoff, especially since the precomputation time is a little smaller too ($2^{5N/8}$ compared to $2^{2N/3}$).

5 Experimental results

We have successfully carried on this attack on some example generators. We give here the timing of our program in C. All computations were performed on a 3.6GHz Pentium4 with 2MB of cache and 2GB of RAM.

We worked on three filter generators of length 53, 59 and 61. In all three cases, the filtering functions used were good cryptographic functions with a maximum Walsh coefficients of respectively 32, 24 and 48. We can see in Table 5 the exact value of the bias for these functions. Notice that it is significantly higher than our lower bound. As a comparison, for a Δ equal to one half and an 8-variable function, our lower bound gives 0.002 for weight 5 and 0.000008 for weight 7.

N	n	bias for weight 5	bias for weight 7
53	8	0.0039	0.000061
59	8	0.0027	0.000021
61	9	0.0014	0.000006

Table 1. Exact bias to detect the zero inputs for the used functions.

In Table 5 we can see the timings for some successful attacks. We can see that the online time is really short and that all the computational effort is spent on computing the low weight multiples. We only used weight 5 multiples because we did not have the 2 weeks time needed to precompute enough weight 7 multiples. In all the attacks, the value of L was just chosen to have a very high probability to get enough zero inputs for f in a keystream of length L.

For the first two filter generators, we applied the exact method described in this paper. For the last attack however, we did not want to spend too much time on the multiples computation, so we used a few tricks to improve the practical

N	n	multiples weight	$\log_2 D$	nb of multiples(time)	L	online time
53	8	5	18.6	100000(20min)	3200	10 sec
59	8	5	20.47	330000(1day)	3200	30 sec
61	9	5	21	349034(2days)	4000	1 min

Table 2. Successful attack timing on the different generators. Some tricks were used for the last generator as explained below.

efficiency. Firstly, at the price of doubling the needed keystream, we can get for each multiple $w+1$ parity check equations (by shifting the multiple by one of its five non null positions). The other improvement is, as explained before, to deal with erroneous zero inputs detection by spending more time on the last phase of the attack. Here for instance, we got 3 erroneous positions among the 10 with the higher $P_{\mathbf{x}}$. To get the correct key, we thus had to try all the $\binom{10}{7}$ linear systems since $\lceil N/n \rceil$ is equal to 7 here. Those tricks helped to perform the attack with less multiples, however we did not really cut down the overall attack complexity.

To conclude this section, notice that the actual value for D is really close to the theoretical one. In order to detect the bias, we theoretically needed respectively around 65746, 137200 and 510000 parity checks for each generator. That gives us, using the approximated formula (17) for the number of multiples, a theoretical $\log_2 D$ of 18.4, 20.16 and 21.14 respectively.

6 Conclusion

As a conclusion, we want to detail some important points about the attack we just presented.

First of all, the probabilistic hypothesis behind the complexity analysis seems quite sound since the simulations are really close to the predicted results. This is actually not always the case with other attacks using a binary symmetric channel model where simulations are usually worse than predicted.

Then, we believe that this attack is difficult to avoid. Using a filtering function with a bad autocorrelation will certainly weaken the cipher. Moreover, in this case other inputs than the all zero one could become detectable. Remark as well that one cannot have a filtering function with too many variables compared to N. This is for performance reasons but also to have tapping positions with good behavior.

Finally, the overall complexity of the attack is quite good as explained at the end of Section 4. In particular, we successfully attacked a length 61 filter generator in a few seconds after a 2 days precomputation on a single computer.

Acknowledgment

The author want to thanks Yann Laigle-Chapuy, Jean-Pierre Tillich and Anne Canteaut for their helpful insight on the subject.

References

[BS00] Alex Biryukov and Adi Shamir. Cryptanalytic time/memory/data tradeoffs for stream ciphers. *Lecture Notes in Computer Science*, 1976, 2000.

[CF02] A. Canteaut and E. Filiol. On the influence of the filtering function on the performance of fast correlation attacks on filter generators. In *23rd Symposium on Information Theory in the Benelux*, Louvain-la-Neuve, Belgium, May 2002.

[CJM02] P. Chose, A. Joux, and M. Mitton. Fast correlation attacks: an algorithmic point of view. In *Advances in Cryptology - EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 209–221. Springer-Verlag, 2002.

[CM03] Nicolas Courtois and Willi Meier. Algebraic attacks on stream ciphers with linear feedback. *Advances in Cryptology – EUROCRYPT 2003*, LNCS 2656:346–359, 2003.

[Cou03] Nicolas Courtois. Fast algebraic attacks on stream ciphers with linear feedback. In *Advances in Cryptology-CRYPTO 2003*, volume 2729 of *LNCS*, pages 176–194. Springer Verlag, 2003.

[CT00] Anne Canteaut and Michaël Trabbia. Improved fast correlation attacks using parity-check equations of weight 4 and 5. In *EUROCRYPT*, pages 573–588, 2000.

[EJ04] Hakan Englund and Thomas Johansson. A new simple technique to attack filter generators and related ciphers. *Selected Areas in Cryptography*, LNCS 3357:39–53, 2004.

[GCD00] Jovan Dj. Golic, Andrew Clark, and Ed Dawson. Generalized inversion attack on nonlinear filter generators. *IEEE Trans. Comput.*, 49(10):1100–1109, 2000.

[GH05] Jovan Dj. Golic and Philip Hawkes. Vectorial approach to fast correlation attacks. *Des. Codes Cryptography*, 35(1):5–19, 2005.

[Gol96] Jovan Dj. Golic. On the security of nonlinear filter generators. In *Proceedings of the Third International Workshop on Fast Software Encryption*, pages 173–188, London, UK, 1996. Springer-Verlag.

[JJ00] Thomas Johansson and Fredrik Jöhansson. Fast correlation attacks through reconstruction of linear polynomials. In *CRYPTO '00: Proceedings of the 20th Annual International Cryptology Conference on Advances in Cryptology*, pages 300–315, London, UK, 2000. Springer-Verlag.

[JJ02] F. Jönsson and T. Johansson. A fast correlation attack on LILI-128. *Information Processing Letters*, 81(3):127–132, February 2002.

[LCPP96] Sangjin Lee, Seongtaek Chee, Sang-Joon Park, and Sung-Mo Park. Conditional correlation attack on nonlinear filter generators. In *ASIACRYPT '96: Proceedings of the International Conference on the Theory and Applications of Cryptology and Information Security*, pages 360–367, London, UK, 1996. Springer-Verlag.

[Lev04a] Sabine Leveiller. A new algorithm for cryptanalysis of filtered lfsrs: the "probability-matching" algorithm. In *ISIT*, volume 1978, pages 234–, 2004.

[Lev04b] Sabine Leveiller. *Quelques algorithmes de cryptanalyse du registre filtré*. PhD thesis, Télécom Paris, ENST, November 2004.

[LZGB03] Sabine Leveiller, Gilles Zémor, Philippe Guillot, and Joseph Boutros. A new cryptanalytic attack for PN-generators filtered by a boolean function. In *SAC '02: Revised Papers from the 9th Annual International Workshop on*

Selected Areas in Cryptography, pages 232–249, London, UK, 2003. Springer-Verlag.

[MFI01] Miodrag J. Mihaljevic, Marc P. C. Fossorier, and Hideki Imai. A low-complexity and high-performance algorithm for the fast correlation attack. *Lecture Notes in Computer Science*, 1978:45–60, 2001.

[MFI05] Miodrag J. Mihaljevic, Marc P. C. Fossorier, and Hideki Imai. Cryptanalysis of keystream generator by decimated sample based algebraic and fast correlation attacks. In *INDOCRYPT*, pages 155–168, 2005.

[MFI06] Miodrag J. Mihaljevic, Marc P. C. Fossorier, and Hideki Imai. A general formulation of algebraic and fast correlation attacks based on dedicated sample decimation. In *AAECC*, pages 203–214, 2006.

[MS88] W. Meier and O. Staffelbach. Fast correlation attacks on stream ciphers. In *Advances in Cryptology - EUROCRYPT'88*, volume 330 of *Lecture Notes in Computer Science*, pages 301–314. Springer-Verlag, 1988.

[RH07] S. Rønjom and T. Helleseth. A new attack on the filter generator. *to appears in IEEE IT*, 2007.

[Sie85] Thomas Siegenthaler. Decrypting a class of stream ciphers using ciphertext only. *IEEE Trans. Computers*, 34(1):81–85, 1985.

A Lemma used in the bias computation

The proof at the end of Section 3 is based on the following lemma applied with $m = 2^{n-1}$, $s = \Delta$ and $a_i = \left(\frac{\widehat{d_1}(\mathbf{u})}{2^n}\right)^2$ for the $\mathbf{u}$ in $\mathbf{F}_2^n$ such that $\mathbf{x}.\mathbf{u} = 1$.

Lemma 1. *Given m positive real numbers $(a_i)_{i=1\ldots m}$ and an integer $p > 1$ we have the lower bound*

$$\sum_{i=1}^{m} {a_i}^p \geq m \left(\frac{s}{m}\right)^p \tag{25}$$

where $s \stackrel{\text{def}}{=} \sum_{i=1}^{m} a_i$.

Proof. The result comes almost directly from Hölder's inequality

$$\sum_{i=1}^{m} |x_i y_i| \leq \left(\sum_{i=1}^{m} |x_i|^p\right)^{1/p} \left(\sum_{i=1}^{m} |y_i|^q\right)^{1/q} \tag{26}$$

where $(x_i)_{i=1\ldots m}$, $(y_i)_{i=1\ldots m}$, p, q are in $\mathbb{R}$ and such that $\frac{1}{p} + \frac{1}{q} = 1$. If we apply it with all the y_i equal to 1, the x_i equal to the positive a_i, the p from the lemma and the corresponding q we obtain

$$\sum_{i=1}^{m} a_i \leq \left(\sum_{i=1}^{m} {a_i}^p\right)^{1/p} m^{1/q} \quad \text{that is} \quad \left(\sum_{i=1}^{m} {a_i}^p\right) \geq (s m^q)^p \,. \tag{27}$$

And since $\frac{1}{q} = \frac{1-p}{p}$ we finally obtain

$$\left(\sum_{i=1}^{m} {a_i}^p\right) \geq s^p m^{1-p} \geq m \left(\frac{s}{m}\right)^p . \tag{28}$$

Remark that there is an equality when all the a_i are equal to s/m.