Finding low-weight polynomial multiples using discrete logarithm

Frédéric Didier, Yann Laigle-Chapuy

Finding low-weight polynomial multiples using discrete logarithm

Frédéric Didier
INRIA Rocquencourt
Projet CODES,
Domaine de Voluceau
78153 le Chesnay cedex
Frederic.Didier@inria.fr

Yann Laigle-Chapuy
INRIA Rocquencourt
Projet CODES,
Domaine de Voluceau
78153 le Chesnay cedex
Yann.Laigle-Chapuy@inria.fr

***Abstract*— Finding low-weight multiples of a binary polynomial is a difficult problem arising in the context of stream ciphers cryptanalysis. The best algorithms to solve this problem are based on a time memory trade-off. Staying in this category, we will present a new approach using discrete logarithm rather than a direct representation of the involved polynomials. This provides an alternative to the previously known algorithms which improves in some case the computational complexity.** [1]

I. Introduction

Correlation and fast correlation attacks are probably the most important classes of attacks against stream ciphers based on linear feedback shift registers (LFSRs). They were originally proposed by Siegenthaler [13] and improved by Meier and Staffelbach [10]. Since then, many different versions have been proposed [1], [8], [7], [9], either very general or adapted to specific designs.

The basic idea is to consider that the output of the stream cipher is a noisy version of a sequence generated by an LFSR with the same initial state. The attack can be seen as an error-correction problem: recover the sequence, and therefore the initial state of the LFSR. To do this most of the attacks take advantage of parity check equations existing in the sequence we are trying to recover. Those parity check equations are in fact given by the multiples of the feedback polynomial, and to keep the bias as low as possible, low-weight multiples are necessary. As a precomputation step, we thus have to find those parity check equations before using them in the active part of the attack.

Depending on our objectives (finding one or many such multiples) and on the parameters (degree of the feedback polynomial and of the multiples, expected weight), there exists different algorithms to find low-weight multiples (see [2], [5]). We will complete them by another approach based on the use of discrete logarithm over finite fields. This will lead to a new algorithm for the computation of polynomials multiples that has better performance for some problems. Remark that the complexity of the best method is often still very high for parameters used in real cryptosystem. Notice also that in [11] discrete logarithms were already used to compute multiples of weight 3 and 4. We have generalized this idea and improved the complexity analysis.

The paper is organized as follows. Section II introduces some notations. The usual approach used to compute low-weight multiples is presented in Section III. In Section IV, we detail our main algorithm and compare its complexity with the algorithm of [2]. Then, we will see in Section V how the complexity is modified when we only want to find a few multiples and not all. Finally, we will discuss in Section VI some important practical points and give some experimental results in Section VII.

II. Preliminary

A. Notations

The problem we will be dealing with is the following.

Problem *1 (Low-weight polynomial multiple):*
Input: A binary primitive polynomial $P \in \mathbf{F_2}[X]$ of degree n, and two integers w and D.
Output: *All* the multiples of P of weight at most w and degree at most D.

The number of expected such multiples of P is heuristically approximated by $\frac{D^{w-1}}{(w-1)!2^n}$, considering that for D large enough, the values of the polynomials of weight w and degree at most D are uniformly distributed.

[1]This work is partially funded by CELAR/DGA.

Most of the time, the degree D and the weight are chosen high enough for many solutions to exist as we need many parity check equations to mount an attack.

It's also worth noticing that we almost never need all the multiples. In fact, to mount a successful attack, one only have to find a fixed number of parity check equations. It is thus sufficient to find many — but not all — multiples, which might be much easier, especially if the constraint on the degree and the weight are high enough. We therefore introduce a slightly different problem.

Problem *2:*

Input: A binary primitive polynomial $P \in \mathbf{F_2}[X]$ of degree n, and three integers w, D and B.

Output: B multiples of P of weight at most w and degree at most D, or as much as possible if there are not B such multiples.

III. The classical approach

A. The algorithm

The main idea is to use a time-memory trade-off (TMTO). Set $w = q_1 + q_2 + 1$ with $q_1 \leq q_2$.

Algorithm *1 (TMTO):*

- For all the q_1-tuples $\Gamma = (\gamma_1, \ldots, \gamma_{q_1})$ with $0 < \gamma_1 < \cdots < \gamma_{q_1} \leq D$, compute and store the pairs $\langle X^{\gamma_1} + \cdots + X^{\gamma_{q_1}} \mod P; \Gamma \rangle$.
- For all q_2-tuples $\Delta = (\delta_1, \ldots, \delta_{q_2})$ with $0 < \delta_1 < \cdots < \delta_{q_2} \leq D$, compute $X^{\delta_1} + \cdots + X^{\delta_{q_2}} \mod P$. Look in the table for an element XORing to 1 (this can be efficiently done by using an hash table). If it exists, this gives
$$1 + \sum_{\gamma \in \Gamma} X^{\gamma} + \sum_{\delta \in \Delta} X^{\delta} = 0 \mod P.$$

B. Complexity

The usual time-memory trade-off is $q_1 = \left\lfloor \frac{w-1}{2} \right\rfloor$ and $q_2 = \left\lceil \frac{w-1}{2} \right\rceil$, in order to balance the complexity of the two phases of the algorithm. The most time consuming part depends on the parity of w, as we do not have to compute anything to find the collisions if $q_1 = q_2$.

The memory complexity is then $\mathcal{O}\left(D^{q_1}\right)$ (for the first phase) while the time complexity is $\mathcal{O}\left(D^{q_2}\right)$. Remark that in [2] the memory usage of the algorithm has been improved in order to use only $\mathcal{O}\left(D^{\lceil \frac{w-1}{4} \rceil}\right)$ bits.

IV. Using discrete logarithm

A. The algorithm

In this section, we will consider the field $\mathbf{F_{2^n}}$ defined as $\mathbf{F_2}[x]/\langle P \rangle$. The discrete logarithm (with base element x) in this field will be denoted by Log.

Set $w = q_1 + q_2 + 2$ with $q_1 \leq q_2$. Take two tuples

$$\Gamma = (\gamma_1, \ldots, \gamma_{q_1}) \text{ with } 0 < \gamma_1 < \cdots < \gamma_{q_1} \leq D$$

and

$$\Delta = (\delta_1, \ldots, \delta_{q_2}) \text{ with } 0 < \delta_1 < \cdots < \delta_{q_2} \leq D.$$

Denoting by L_Γ and L_Δ the logarithms of $1 + \sum_{\gamma \in \Gamma} x^{\gamma}$ and $1 + \sum_{\delta \in \Delta} x^{\delta}$ respectively, the following equalities hold in $\mathbf{F_2}[x]/\langle P \rangle$:

$$1 + \sum_{\gamma \in \Gamma} x^{\gamma} = x^{L_\Gamma - L_\Delta} \left(1 + \sum_{\delta \in \Delta} x^{\delta}\right) \quad \text{and}$$

$$x^{L_\Delta - L_\Gamma} \left(1 + \sum_{\gamma \in \Gamma} x^{\gamma}\right) = 1 + \sum_{\delta \in \Delta} x^{\delta}.$$

Now let $e \in \ ]-2^{n-1}, 2^{n-1}]$ such that e is equal to $L_\Gamma - L_\Delta$ modulo $2^n - 1$. If $e > 0$, then the polynomial

$$\left(1 + \sum_{\gamma \in \Gamma} x^{\gamma}\right) + x^{e} \left(1 + \sum_{\delta \in \Delta} x^{\delta}\right) \tag{1}$$

is a multiple of P with degree $\max(\gamma_{q_1}, \delta_{q_2} + e)$. If $e < 0$, then the polynomial

$$x^{-e} \left(1 + \sum_{\gamma \in \Gamma} x^{\gamma}\right) + \left(1 + \sum_{\delta \in \Delta} x^{\delta}\right) \tag{2}$$

is a multiple of P with degree $\max(\gamma_{q_1} - e, \delta_{q_2})$. So, if one of the two following conditions is satisfied

$$\begin{aligned} e > 0 \quad &\text{and} \quad \delta_{q_2} + e \leq D \\ e < 0 \quad &\text{and} \quad \gamma_{q_1} - e \leq D \end{aligned}$$

we get a multiple of P with degree at most D and weight at most w. We can rewrite both conditions in a single inequality

$$\gamma_{q_1} - D \leq e \leq D - \delta_{q_2}. \tag{3}$$

The algorithm is then straightforward.

Algorithm *2 (LogTMTO):*

- For all the q_1-tuples $\Gamma = (\gamma_1, \ldots, \gamma_{q_1})$ with $0 < \gamma_1 < \cdots < \gamma_{q_1} \leq D$, compute
$$L_\Gamma = Log\left(1 + x^{\gamma_1} + \cdots + x^{\gamma_{q_1}}\right)$$
and store the pairs $\langle L_\Gamma; \Gamma \rangle$.
- For all q_2-tuples $\Delta = (\delta_1, \ldots, \delta_{q_2})$ with $0 < \delta_1 < \cdots < \delta_{q_2} \leq D$ compute the logarithm
$$L_\Delta = Log\left(1 + x^{\delta_1} + \cdots + x^{\delta_{q_2}}\right)$$

and look in the table for all the elements with a logarithm L_Γ satisfying (3). For each of them we obtain a multiple of P given by (1) or (2) depending on the sign of e.

Of course, since we can decompose all polynomials of weight w in $\binom{w-1}{q_1}$ way, we obtain each multiple many times.

B. Complexity

In order to perform the second phase, one could sort the table with increasing logarithms, but using an appropriate data structure like an hash table indexed by the most significants bits of the logarithm is a lot more efficient. As long as $D < 2^{n/2}$, the search cost is $\mathcal{O}(1)$.

Once again, we choose the parameters of the time-memory trade-off in order to balance the complexity of the two phases, taking $q_1 = \left\lfloor \frac{w-2}{2} \right\rfloor$ and $q_2 = \left\lceil \frac{w-2}{2} \right\rceil$.

As for the classical algorithm, the most time consuming part depends on the parity of w as we do not have to compute any logarithm in the second phase if $q_1 = q_2$.

The memory usage is then $\mathcal{O}(D^{q_1})$, while the time complexity is $\mathcal{O}(D^{q_2})$ logarithm computations. We will see in Section VI-B that the logarithm can be computed quite efficiently. Actually for many practical values of n we can even compute it in $\mathcal{O}(1)$. Hence we neglect it in Table I.

TABLE I
COMPARISON BETWEEN TMTO AND LOGTMTO

	$w = 2p$		$w = 2p+1$	
Algorithm	Time	Memory	Time	Memory
TMTO	D^p	$D^{\lfloor p/2 \rfloor}$	D^p	$D^{\lfloor p/2 \rfloor}$
LogTMTO	D^{p-1}	D^{p-1}	D^p	D^{p-1}

As we can see in Table I, if w is even we can improve the time complexity compared to the classical approach. Heuristically, the improvement by a factor D can be explained by the fact that we look for values in an interval of size roughly D instead of exact collisions.

Regarding the memory however, as explained in [2] the computation behind the classical algorithm can be done using only $\mathcal{O}\left(D^{\lceil p/2 \rceil}\right)$ bits. So the discrete logarithms approach is always worse for odd w and will only be of practical interest when we are looking for all the multiples of weight 4 and maybe 6. After that, the memory usage just become too important.

However, we will see in the next section that when we are only looking for a small fraction of all the multiples of degree up to D, the discrete logarithms method can be quite efficient.

V. FIND MANY BUT NOT ALL

We deal in this section with the problem of finding a small proportion of all the multiples of weight w and degree at most D (Problem 2). If the number B of polynomials we want is small enough, depending on the parameters, we can do better than the previous algorithms.

A very basic approach is to try random polynomials of weight w until we actually find a multiple. In expectation we will then find a multiple every 2^n polynomials tried. We can also do the same using discrete logarithms. By computing logarithms for polynomials A of weight $w-1$ and degree less than D, we can obtain easily low-weight multiples of type $A + x^{Log(A)}$ if the logarithm is at most D. The expectation here is to find a multiple every $2^n/D$ iterations and we have won a factor D.

However, the best methods to solve this problem are once again TMTO. The algorithms are just simple variations of the previous ones when we put the elements in the hash table one by one and stop when we have found enough multiples.

Applying the birthday paradox, we can thus find with the basic algorithm a multiple with a time and memory complexity of $\mathcal{O}\left(\sqrt{2^n}\right)$ in average. Using discrete logarithms, we will find a multiple as soon as two logarithms have a distance by approximately D. The complexity is then in $\mathcal{O}\left(\sqrt{\frac{2^n}{D}}\right)$ both in time and memory. Remark that in this case one cannot use the improvement of [2] to gain memory. There is also another approach based on Wagner's generalized birthday paradox (see [14], [5]) that can be usefull when w is large. Its complexity is in $\mathcal{O}\left(2^a 2^{n/(a+1)}\right)$ for a a such that $\binom{D}{(w-1)/2^a} \geq 2^{n/(a+1)}$.

As a conclusion to this section, when computing logarithms in $\mathbf{F}^\star_{\mathbf{2^n}}$ is easy, we can gain a factor $\sqrt{D}$ in time and memory to find a multiple. Notice also that in practice when we need many multiples, we can design an algorithm between the one that compute all the multiples and the one presented here in order to get the best performance. We will see an illustration of this in Section VII.

VI. PRACTICAL CONSIDERATIONS

A. Bounds on the degree

First of all, it is worth noticing that it is not necessary to compute all the multiples up to the degree D to take all q_2-tuples up to the degree D.

As a polynomial of weight w has many representations as a sum of a polynomial of weight $q_1 + 1$ and $q_2 + 1$

respectively, we can choose the one with the smallest q_2-tuple.

Proposition *1:* Let $M = 1 + \sum_{i \in I} X^i$ be a multiple of P of weight $w = q_1 + q_2 + 2$ and degree at most D.

Then there exists an integer $1 \leq e \leq D$ and two polynomials A and B of respective weight q_1 and q_2 and of degree respectively at most D and at most $\frac{Dq_2}{w-1}$ such that $M = (1 + A) + X^e (1 + B)$ or $X^e (1 + A) + (1 + B)$.

With the usual trade-off, we can restrict ourselves to the degree $D/2$, dividing the cost of the second phase approximately by a factor $2^{w/2}$.

B. How to compute logarithms

In practice, it is important to compute efficiently discrete logarithms in $\mathbf{F}_{\mathbf{2^n}}^\star$ and hopefully there exists well studied algorithms to do that. It is important to take into account that we are going to compute many logarithms and not only one. All the efficient algorithms for computing logarithms (Baby-step Giant-step, Pohlig-Hellman algorithm [12] and Coppersmith algorithm [3], [4]) can profit from a bigger precomputation step that can be done once and for all. For instance, if $2^n - 1$ is smooth enough, one can tabulate the logarithms in all the subgroups of $\mathbf{F}_{\mathbf{2^n}}^\star$ to make the Pohlig-Hellman algorithm very efficient. In this case, a subsequent discrete logarithm computation can be done in $\mathcal{O}(1)$. This approach can be used for all the n up to 78 except $\{37, 41, 49, 59, 61, 62, 65, 67, 69, 71, 74, 77\}$. In addition we have listed in Table II some larger n for which it is applicable and the corresponding memory requirement. Notice that a full tabulation corresponds to a Giant-step of 1 and that by increasing a little this Giant-step, we can efficiently deal with more values of n .

TABLE II
MEMORY USAGE FOR A FULLY TABULATED POHLIG-HELLMAN ALGORITHM AND SOME SMOOTH $2^n - 1$

n	53	96	110	156	210
memory	439MB	510MB	1.7GB	940MB	201MB

This leads to a very easy and efficient implementation as we will see in Section VII. Moreover, for the most useful cases (that is $w \in \{3, 4, 5\}$) we have to compute logarithms of the form $Log\left(1 + x^i\right)$. This logarithm is know as the Zech's logarithm of i, and we can exploit some properties of Zech's logarithm (see [6]) to speed up the computation. Actually, by computing one Zech logarithm we get $6n$ other logarithms for free. Of course not all of them are useful for us, but the computation time can be divided by a factor of at least 2.

VII. EXPERIMENTAL RESULT

We have implemented our algorithm in C to test its efficiency. The computer used for our experiments is a 3.6GHz Pentium4 with 2MB of cache and 2GB of RAM.

A. Problem 1

We give in Table III the timings to find all the multiples of weight w up to degree D of the polynomial $P = x^{53}+x^{47}+x^{45}+x^{44}+x^{42}+x^{40}+x^{39}+x^{38}+x^{36}+x^{33}+x^{32}+x^{31}+x^{30}+x^{28}+x^{27}+x^{26}+x^{25}+x^{21}+x^{20}+x^{17}+x^{16}+x^{15}+x^{13}+x^{11}+x^{10}+x^7+x^6+x^3+x^2+x^1+1$.

As explained in the previous section, we used a fully tabulated Pohlig-Hellman.

TABLE III
PROBLEM 1: FIND ALL THE MULTIPLES UP TO DEGREE D

n	53					
w	4			5		
$\log_2(D)$	20	22	28	13	14	16
time	$47''$	$2'02''$	$1h52'$	$4'11''$	$14'40''$	$3h33'$

We can see that the algorithm is, as expected, very efficient for weight 4 as its complexity is linear in the degree D, both for time and memory (to be compared to a quadratic complexity for the classical approach).

We were also able to compute all the multiples of weight 5 and degree up to 2^{16} of a polynomial of degree 53 within a few hours. But for the degree 5 the algorithm of [2] is more efficient.

B. Problem 2

With the same polynomial of degree $n = 53$, we also looked for multiples with an higher weight $w = 7$, and degree at most $D = 2^{15}$. In order to do that, we precomputed all the trinomials $(1 + x^{\gamma_1} + x^{\gamma_2})$ up to the degree K, which corresponds to $q_1 = 2$, instead of 3 for the optimal trade-off. We then computed many discrete logarithm of random polynomials $(1+x^{\delta_1}+x^{\delta_2}+x^{\delta_3}+x^{\delta_4})$ in order to find multiples of weight 7. The results are given in Figure 1 where we see that a bigger precomputation can greatly improve the performance.

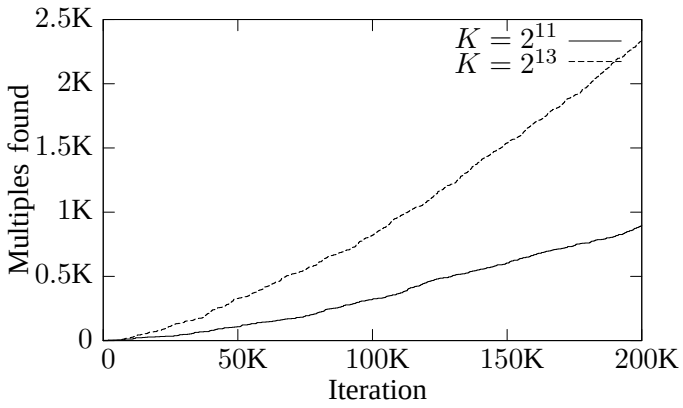

Fig. 1. Evolution of the number of multiples of weight 7 and degree lower than 2^{15} found with precomputed logarithms up to degree K

VIII. Conclusion

In this paper, we devised an algorithm to find low-weight multiples of a given binary polynomial that appears to be efficient for two cases that actually occur in practice.

The first case is when we are looking for all the multiples of weight 4 and degree at most D of a given polynomial of degree n. The complexity is then in $\mathcal{O}(D)$ discrete logarithms computation in $\mathbf{F}^{\star}_{\mathbf{2^n}}$ where the other approach run in $\mathcal{O}\left(D^2\right)$. So the best algorithm will depends on the complexity of a discrete logarithm computation in $\mathbf{F}^{\star}_{\mathbf{2^n}}$ which can be smaller than D in many practical situations. Notice that our algorithm may also give better performance for multiples of weight 6.

The other case where discrete logarithms can be useful is when we are only looking for a small fraction of all the possible multiples. The complexity to find one of them is then $\mathcal{O}\left(\sqrt{\frac{2^n}{D}}\right)$ logarithm computations.

Acknowledgment

The authors would like to thank Anne Canteaut and Jean-Pierre Tillich for their helpful insights on the subject.

References

[1] A. Canteaut and M. Trabbia. Improved fast correlation attacks using parity-check equations of weight 4 and 5. In *Advances in Cryptology - EUROCRYPT'2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 573–588. Springer-Verlag, 2000.

[2] 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.

[3] Don Coppersmith. Evaluating logarithms in GF(2n). In *STOC '84: Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 201–207, New York, NY, USA, 1984. ACM Press.

[4] Don Coppersmith. Fast evaluation of logarithms in fields of characteristic two. *IEEE Transactions on Information Theory*, 30(4):587–593, 1984.

[5] Matthieu Finiasz and Serge Vaudenay. When stream cipher analysis meets public-key cryptography. In E. Biham and A. Youssef, editors, *SAC 2006*, Lecture Notes in Computer Science. Springer, 2006.

[6] Klaus Huber. Some comments on zech's logarithms. *IEEE Transactions on Information Theory*, 36(4):946–, 1990.

[7] T. Johansson and F. Jönsson. Fast correlation attacks based on turbo code techniques. In *Advances in Cryptology - CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 181–197. Springer-Verlag, 1999.

[8] T. Johansson and F. Jönsson. Improved fast correlation attack on stream ciphers via convolutional codes. In *Advances in Cryptology - EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 347–362. Springer-Verlag, 1999.

[9] T. Johansson and F. Jönsson. Fast correlation attacks through reconstruction of linear polynomials. In *Advances in Cryptology - CRYPTO'00*, volume 1880 of *Lecture Notes in Computer Science*, pages 300–315. Springer-Verlag, 2000.

[10] 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.

[11] W.T. Penzhorn and G.J. Kühn. Computation of low-weight parity checks for correlation attacks on stream ciphers. In *Cryptography and Coding - 5th IMA Conference*, volume 1025 of *Lecture Notes in Computer Science*, pages 74–83. Springer-Verlag, 1995.

[12] S.C. Pohlig and M.E. Hellman. An improved algorithm for computing logarithms over GF(p) and its cryptographic significance. *IEEE Transactions on Information Theory*, IT-24:106–110, 1978.

[13] Thomas Siegenthaler. Cryptanalysts representation of nonlinearly filtered ml-sequences. In *EUROCRYPT*, pages 103–110, 1985.

[14] David Wagner. A generalized birthday problem. In *Advances in cryptology—CRYPTO 2002*, volume 2442 of *Lecture Notes in Comput. Sci.*, pages 288–303. Springer, Berlin, 2002.