Finding Low Weight Polynomial Multiples Using Lattices

Laila El Aimani and Joachim von zur Gathen

B-IT, Dahlmannstr. 2, Universität Bonn, 53113 Bonn, Germany
elaimani,gathen@bit.uni-bonn.de

Abstract. The low weight polynomial multiple problem arises in the context of stream ciphers cryptanalysis and of efficient finite field arithmetic, and is believed to be difficult. It can be formulated as follows: given a polynomial $f \in \mathbb{F}_2[X]$ of degree d, and a bound n, the task is to find a low weight multiple of f of degree at most n. The best algorithm known so far to solve this problem is based on a time memory trade-off and runs in time $\mathcal{O}(n^{\lceil (w-1)/2 \rceil})$ using $\mathcal{O}(n^{\lceil (w-1)/4 \rceil})$ of memory, where w is the estimated minimal weight. In this paper, we propose a new technique to find low weight multiples using lattice basis reduction. Our algorithm runs in time $\mathcal{O}(n(n-d)^5)$ and uses $\mathcal{O}(nd)$ of memory. This improves the space needed and gives a better theoretical time estimate when $w \geq 12$ or when the *excess degree* $n-d$ is small, say, $(n-d)^5 < n^{\lceil (w-3)/2 \rceil}$. The former situation is plausible when the bound n, which represents the available keystream, is small, whereas the latter one occurs in efficient finite field arithmetic. We also propose bounds for the minimal weight of such multiples, supplying in this sense the state-of-the art techniques with a method to check whether their estimated minimal weight is in the correct range. This provides a quantitative cryptographic quality criterion for such polynomials: the fewer low degree low weight multiples a polynomial has, the harder becomes this type of cryptanalysis of the corresponding stream cipher. As an example, the Bluetooth polynomial turns out to be of good quality in this sense. Moreover, we introduce the corresponding number problem and apply a similar strategy to find sparse multiples of a given number with respect to the Hamming weight of their 2-ary representation. Finally, we run our experiments using the NTL library on some known polynomials in cryptanalysis and we confirm our analysis.
Keywords: stream ciphers analysis, low weight polynomial multiples, lattices, shortest vector.

1 Introduction

Finding a low weight multiple of a polynomial over $\mathbb{F}_2$ is believed to be a difficult [1] problem. In fact, there exists no known polynomial time algorithm to solve it. Later in this document we point out a reduction from this problem to the Syndrome Decoding problem which is known to be NP-complete, however the other direction hasn't been investigated to the best of the authors' knowledge. Anyway, the presumed difficulty of the problem has motivated Finiasz and Vaudenay [9] to propose a public key cryptosystem whose security rests on the intractability of this problem.

The problem can formulated as follows, given a polynomial f over a finite field, $\mathbb{F}_2$ for instance, and a bound n, determine the set:

$$M_f(n,w) = \{g \in \mathbb{F}_2[X] \colon f | g, \deg(g) < n, \text{weight}(g) \leq w\},$$

where w is the least possible weight: $w = \min\{w_i \colon M_f(n,w_i) \neq \emptyset\}$. It is often enough to compute sufficiently many - but not all- elements from this set.
There exists also the other variant which consists of determining the set $M_f(n,w)$ for a given weight w and for $n = \min\{n_i \colon M_f(n_i,w) \neq \emptyset\}$. In this paper, we concentrate on the first variant.

The low weight polynomial multiple problem originated in cryptography from two distinct areas: attacks on LFSR-based stream ciphers and efficient finite field arithmetic.

[1] Oppositely to its *inverse* problem which lies in finding factors of low weight polynomials, for instance trinomials, and for which there exist efficient algorithms (Brent et al. [3]).

Application to stream ciphers cryptanalysis

Stream ciphers constitute an important class of secret-key encryption algorithms. In fact, LFSR-based stream ciphers are widely used in many applications because of the advantages they present compared to other encryption schemes, for instance, block ciphers: they are faster, require less hardware circuitry and have fewer propagation errors. An example is Bluetooth encryption. Stream ciphers consist of a seed, corresponding to the shared secret key, and a pseudorandom generator, which consists of constituent LFSRs [17] and a nonlinear combination function. The result is a pseudo-random binary sequence, called the *keystream*, which is, in the case of a binary additive stream cipher, bitwise added to the plaintext in order to obtain the ciphertext. Hence, attacks on stream ciphers have as ultimate goal the recovery of the initializations of the LFSRs. *Correlation attacks* are considered to be the most important class of attacks against stream ciphers. There exists also a category of attacks that simply aim at verifying whether a bitstream is the encryption of some (unknown) message, the so-called *distinguishing attacks*. Both attacks require finding low weight multiples of a constituent LFSR's feedback polynomial.

Fast correlation attacks. They were originally introduced by Siegenthaler [27] and later improved by Meier and Staffelbach [16]. Since then, a series of proposals sprang up, either very general or adapted to a specific scheme, to name but a few [12, 11, 13, 4, 5]. The principle of this type of attacks is as follows: we try to reconstruct the initialization of the constituent LFSR, say the i-th one, from the output keystream by viewing the latter as the transmission of the former one through a noisy channel. In fact, we assume that the adversary knows both the plaintext and the ciphertext (a known plaintext attack). The errors resulting from this transmission are due to the other registers. Let s and s^i denote the output of the keystream generator and the i-th LFSR R_i respectively. The more the sequences s and s^i, are correlated, the smaller is the attack's error probability. More precisely, let $s^i = (s^i_0, \ldots, s^i_{N-1})$ be the initial N-bit sequence generated by the constituent LFSR R_i whose connection polynomial is f with linear complexity L, and $s = (s_0, \ldots, s_{N-1})$ be the initial N-bit keystream. Let further $p = \text{Prob}(s^i_k = s_k)$ be the correlation probability between s and s^i, where the probability is taken over the possible initializations of the constituent LFSRs. Then s can be viewed as the result of the transmission of s^i through a binary symmetric channel with error probability $1 - p$. Moreover, the sequence s^i satisfies the linear recurrence defined by the polynomial f. Thus the word $s^i = (s^i_0, \ldots, s^i_{N-1})$ belongs to the linear error correcting code of length N and of dimension L defined by f. We can then recover it using the iterative decoding process due to Gallager [10] which exploits the existence of parity check equations.
Fast correlation attacks can then be mounted into two phases: the first one determines low weight parity check equations or equivalently low weight multiples of an LFSR's connection polynomial, whereas the second phase decodes the sequence s to recover s^i. R_i could then be recovered as soon as $N \geq L$.

Distinguishing attacks. A distinguishing attack as previously stated can be used to verify or falsify whether a bitstream is the encryption of some message. This is of significant importance if the set of possible messages or possible keys is small. In fact, a small message set gives few possibilities for the keystream, this could be obtained by bitwise adding the given ciphertext to the possible messages. Then, one can simply check the correct keystream by encrypting some known bitstreams using the possible keystreams and feeding the resulting ciphertexts to the distinguisher, the correct keystream is the one providing a ciphertext that is identified by the distinguisher as yes instance. In case the key size is small such that an exhaustive search is plausible, distinguishing attacks are then equivalent to key-recovery attacks and thus could be

employed to decrypt the ciphertexts.

Low weight multiple polynomials are also required in such attacks, in fact, following the framework described in the above paragraph, namely, an LFSR-based stream cipher given by constituent LFSRs and a pseudo-random generator. We assume that the output keystream s is written as the sum of a binary biased sequence b, i.e., a sequence such that $\text{Prob}(b_i = 0) = 1/2 + \gamma$, $\gamma > 0$[2], and an LFSR's output l (could be the equivalent LFSR of a subset of the constituent LFSRs combined via a nonlinear function). Let $M = \sum_{i=1}^{w} X^{q_i}$ be a multiple of the LFSR's connection polynomial of degree n and weight w, where $0 = q_1 < q_2 < \ldots < q_w = n$. Then, by standard cryptanalytic techniques, the output keystream is biased with bias $\frac{1}{2}\gamma^w$, since $\oplus_{i=1}^{w} l_{t+q_i} = 0$ holds for all t and $\oplus_{i=1}^{w} s_{t+q_i} = \oplus_{i=1}^{w}(l_{t+q_i} + b_{t+q_i}) = \oplus_{i=1}^{w} b_{t+q_i}$. It follows that one needs $\gamma^{-2\cdot w}$ samples to distinguish the output keystream from a truly random sequence. Notice, that the smaller w, the higher the bias will be and thus the fewer samples are needed to build the distinguisher. For examples of such attacks, see [15] on E0, and [8] on SOBER-t16 and SOBER-t32.

Application to efficient finite field arithmetic

It is often attractive to use finite fields $\mathbb{F}_{2^n}$ in cryptography, in particular for hardware applications. There are several ways of representing small fields. One representation is by a sparse irreducible polynomial $g \in \mathbb{F}_2[X]$ of degree n, as $\mathbb{F}_{2^n} = \mathbb{F}_2[X]/(g)$. In [28], this was found to be the most efficient representation if exponentiation is a core operation. Ideally, one would like to use the minimal possible weight, that is, trinomials of weight 3. However, these do not always exist. Brent and Zimmermann [2] proposed an interesting solution: take an irreducible polynomial $f \in \mathbb{F}_2[X]$ of degree n, but possibly large weight, a multiple g of f with small weight, say g a trinomial, and work in the ring $R = \mathbb{F}_2[X]/(g)$ most of the time, going back to the field via $R \to \mathbb{F}_{2^n}$ only when necessary. This is particularly useful if the *excess degree* $\deg(g) - \deg(f)$ is small. They actually describe efficient algorithms for finding trinomials with large irreducible (and possibly primitive) factors and give examples of such trinomials. A systematic illustration of this method is in preparation.

Our contributions

The main result of the present paper dwells in a new algorithm to compute sparse multiples, with degrees at most a certain n, for a given polynomial f, over $\mathbb{F}_2$, of degree $d < n$. Our algorithm is a lattice-based solution, i.e., consists of the basis reduction of an $(n-d)$-dimensional lattice in $\mathbb{Z}^n$. Hence, it runs in $\mathcal{O}(n(n-d)^5)$ in case the LLL reduction is applied. This gives a better time estimate compared to the standard techniques which run in $\mathcal{O}(n^{\lceil (w-1)/2 \rceil})$ in case w is big, say bigger than 12, or in case the excess degree $n-d$ is small, namely, $(n-d)^5 < n^{\lceil (w-3)/2 \rceil}$. Furthermore, our solution presents a huge improvement in the space complexity, that is $\mathcal{O}(n \cdot d)$ versus $\mathcal{O}(n^{\lceil (w-1)/4 \rceil})$.
In contrast to our algorithm, the standard techniques estimate first the minimal weight, using a heuristic method, then try to find multiples with weight smaller than the expected weight. This heuristic method is independent of the polynomial, in fact, the parameters that come into play are just the degree of the given polynomial and the bound on the multiple's degree. The heuristic works well for random polynomials. But there exist polynomials for which the method predicts a weight that actually doesn't exist. This leads to run unproductively the algorithm to find the sparse multiples. In fact, it is a goal of the cryptosystems designers to come up with such "hard"

[2] The probability is again taken over the possible initializations of the constituent LFSRs.

polynomials which thwart this type of attacks. This leads to a quantitative quality criterion for such polynomials: the higher the degree of multiples with a given weight, the more resistant they are to such attacks. To overcome this problem, at least partially, we propose theoretical bounds for the "real" weight using known results from lattices. In this way, these techniques will proceed to find the sparse multiples only if the estimated weight is in the correct range. Besides, this result can be parsed from stream ciphers designers in the following way: given a polynomial f (feedback polynomial of a constituent LFSR), they insure that an adversary cannot find a multiple with weight smaller than a given bound given access to a certain amount of keystream. Finally, we introduce the corresponding number problem which consists of finding sparse multiples, with regard to the Hamming weight of the 2-ary representation, of a given number. We apply almost the same strategy based on lattices and confirm our analysis by running the algorithm on some problem instances.
The rest of the paper is organized as follows; first, we give some preliminaries about lattices, for instance we recall the definition of the orthogonal lattice which plays a central role in our algorithm. Second, we present our solution to find sparse multiples for a given polynomial; after giving the approach, we provide experiments as well as comparisons with the state-of-the art techniques in order to confirm our analysis. In section 4, we give bounds for the target weight using some results on lattices. In section 5, we present our algorithm to find sparse binary multiples of a given integer. Finally, we conclude with general thoughts and prospectives.

2 Preliminaries

In this section we give some preliminaries about lattices and their algorithmic problems. The book [18] constitutes a good introduction to this topic.

Let $\mathbb{R}^n$ be the n-dimensional Euclidean space. A lattice L is the set

$$L(b_1, \ldots, b_d) = \{\sum_{i=1}^{d} x_i b_i : x_i \in \mathbb{Z}\},$$

of all integral combinations of d linearly independent vectors (over $\mathbb{R}^n$) $b_1, \ldots, b_d$. Then, d and $B = (b_1, \ldots, b_d)$ are called the **rank** and **basis** of L, respectively.
A lattice L can be generated by more that one basis. These bases, referred to as *equivalent bases* share the same number of elements, called **rank** or **dimension** of the lattice as well as the same **Gram determinant** $\Delta(L) = \Delta(b_1, \ldots, b_d) = \det(G)$, where G is the Gram matrix: $G = (< b_i, b_j >)_{1 \leq i,j \leq d}$ and $< \cdot, \cdot >$ denotes the usual inner product. The **determinant** or **volume** of the lattice, denoted as $\det(L)$, is by definition $\sqrt{\Delta(L)}$.

Definition 1. ***(Successive Minima)*** *Let L be a d-dimensional lattice and let $\mathcal{B}_d(0, r) = \{x \in \mathbb{R}^d : \|x\| < r\}$ be the d-dimensional open ball of radius r centered in 0. The successive minima of L, are constants $\lambda_1(L), \ldots, \lambda_d(L)$ verifying the following: $\lambda_i = \inf\{r : \dim(\mathrm{span}(L \cap \mathcal{B}_d(0, r))) \geq i\}$. We clearly have $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_d$. We call gap of the lattice the ratio between the first and second minima. Finally, the first minimum λ_1 is called also norm of the lattice and corresponds to the norm of the shortest vector in the lattice.*

Theorem 1. *Let B be a d-dimensional lattice basis, and let B^* be the corresponding Gram-Schmidt orthogonalization. Then, the first minimum λ_1 of the lattice (in the ℓ_2-norm) satisfies: $\min_j \|b_j^*\| \leq \lambda_1 \leq \sqrt{\gamma_d}\det(L)^{1/d}$, where γ_d is the Hermite's constant.*

Proof. The lower bound is proved in [18, Basics/Lattices/Succesive minima/Theorem 1.1] whereas the upper bound follows from a classical result of Minkowski which states that for any d-dimensional lattice L and for any $r \leq d$: $\prod_{i=1}^{r} \lambda_i(L) \leq \sqrt{\gamma_d^r}\det(L)^{r/d}$.

□

We get now to the *orthogonal lattice*, a notion which was first introduced in a cryptanalytic context by Nguyen and Stern in 1997 [21]. It has proved very important and was used to attack many public key cryptosystems [21–23].

Definition 2. *(**Orthogonal Lattice**) Let L be a lattice in $\mathbb{Z}^n$, and let* $\mathrm{span}(L)$ *be the vector space (over $\mathbb{R}$) generated by L. The orthogonal lattice is defined as follows:*

$$L^\perp = \mathrm{span}(L)^\perp \cap \mathbb{Z}^n = \{x \in \mathbb{Z}^n : \forall y \in L, < x, y >= 0\}.$$

The biorthogonal $(L^\perp)^\perp$ contains L but generally it is not equal to it. We define the *completed lattice* $\bar{L}$ as being $(L^\perp)^\perp$. It can be viewed as the intersection of $\mathbb{Z}^n$ and $\mathrm{span}(L)$. The determinant of L and $\bar{L}$ are related by the following theorem.

Theorem 2. *Let L be a d-dimensional lattice in $\mathbb{Z}^n$, given by a basis $(b_1, \ldots, b_d)$ and $b_i = (b_i^1, \ldots, b_i^n)$. Then:* $\det(L) = det(\bar{L}) \cdot \prod_{i=1}^{d} \gcd(b_i^1, \ldots, b_i^n)$.

This theorem is quite intuitive and probably already known. Since we have not been able to locate appropriate references, we provide its proof in appendix A.

Moreover, we have the following result [19, Chapter 2/Lemma 2.7 and Theorem 2.8]

Theorem 3. *If L is a lattice in $\mathbb{Z}^n$, then* $\dim(L) + \dim(L^\perp) = n$ *and* $\det(L^\perp) = \det(\bar{L})$.

□

Finally, computing the orthogonal lattice amounts to determining the kernel of a matrix (as a $\mathbb{Z}$-module); see [6, Algorithm 2.4.10]

Theorem 4. *Given a basis of a lattice L in $\mathbb{Z}^n$, one can compute a basis of the orthogonal lattice $L^\perp$ in time polynomial in n, d and the bit-size of the basis.*

□

3 Finding Low Weight Polynomial Multiples Using Lattices

The idea underlying our approach is simple and based on the remark that a low weight polynomial multiple of degree less than n is a low weight linear combination with integer coefficients of the monomials X^i, $0 \leq i < n$, that evaluates to zero modulo the given polynomial. The algorithm follows then in a straightforward way.

Input: a polynomial f of degree d and a bound $n > d$.
Output: $(n - d)$ multiples of f of degree less than n.

1. Compute $h_i = X^i \bmod f$ for all $0 \leq i < n$ and build the $d \times n$ matrix M_n whose columns are the coefficients of the h_i's ;
2. Consider the lattice L_n in $\mathbb{R}^n$ generated by the rows of the matrix M_n ;
3. Compute a basis of the orthogonal lattice $L_n^\perp$ and reduce it ;
4. The $(n - d)$ basis vectors constitute the $(n - d)$ polynomial multiples. For instance, if $v = (v_0, \ldots, v_{n-1})$ is a basis vector, then $m = \sum_{0 \leq i < n} (v_i \bmod 2) X^i$ is a multiple of f ;

Algorithm 1: **Computing low weight multiples of a given polynomial**

The hope is that the multiples computed above are sparse; this is discussed below.

3.1 Analysis

The first two steps can be clearly performed in $\mathcal{O}(d(n-d))$ arithmetic operations. In fact, to compute $h_i = X^i \bmod f$, $0 \leq i < n$, we compute $h_i = X^i$ for $0 \leq i < d$, and for the remaining indices we use the fact that $X^i = Xh_{i-1}$ is either Xh_{i-1} (if $\deg(h_{i-1}) < d-1$) or $Xh_{i-1} + f$ if ($\deg(h_{i-1}) = d-1$). The computation and reduction of the orthogonal lattice basis can be performed in one step as in [21], or in two steps; first compute the integer kernel (as a free $\mathbb{Z}$-module and not as a vector space) of the matrix M_n's transpose using algorithms from [6], then reduce it. In this problem, we can compute the orthogonal lattice $L_n^\perp$ in almost linear time in $(n-d)$ using a simple remark from elementary linear algebra.

Computation of the orthogonal lattice. The lattice $L_n \subseteq \mathbb{R}^n$, has dimension d since the first d components of its generators form a unit matrix, and thus the generators are linearly independent. The orthogonal lattice $L_n^\perp$ then has dimension $n-d$ according to Theorem 3 . Moreover, we can construct this orthogonal lattice incrementally, i.e., from $L_n^\perp$, one can easily derive $L_{n+1}^\perp$. Indeed, let $\mathcal{L} = (l_1, \ldots, l_{n-d})$ be a basis of $L_n^\perp$. It is clear that $(l_i, 0) \in L_{n+1}^\perp$. Let now $m_{i,j}$, where $0 \leq i \leq d-1$ and $0 \leq j \leq n$, be the entries of the matrix M_{n+1}. The first d columns of M_{n+1} correspond to the columns of the identity matrix $I_d \in \mathbb{R}^{d \times d}$. By definition, the other columns $X^j \bmod f$, $d \leq j \leq n$, of M_{n+1} are linear combinations of the the first d columns with coefficients $m_{i,j}$, for instance: $X^n \equiv \sum_{0 \leq i \leq d-1} m_{i,n} X^i \bmod f$. It follows that

$$\sum_{0 \leq i \leq d-1} -m_{i,n} X^i + \sum_{d \leq i \leq n-1} 0 \cdot X^i + X^n \equiv 0 \bmod f, \text{ or } \sum_{0 \leq i \leq d-1} -m_{i,n} h^i + \sum_{d \leq i \leq n-1} 0 \cdot h^i + h^n = 0,$$

where $h_i = X^i \bmod f$. Hence, the vector $u = (-m_{0,n}, \ldots, -m_{d-1,n}, 0, \ldots, 0, 1)$ is also in $L_{n+1}^\perp$ and linearly independent of the vectors $(l_i, 0)$. Since $\dim(L_{n+1}^\perp) = \dim(L_n^\perp) + 1$, we suggest the following: if $\mathcal{L} = (l_1, \ldots, l_{n-d})$ is a basis of $L_n^\perp$ then $\mathcal{K} = (k_1, \ldots, k_{n+1-d})$ is a basis of $L_{n+1}^\perp$ where $k_i = (l_i, 0)$ for $1 \leq i \leq n-d$ and $k_{n+1-d} = u$. We derive then the following algorithm to compute $L_n^\perp$:

Input: The lattice L_n or equivalently the matrix $M_n = (m_{i,j})$, where $0 \leq i < d$ and $0 \leq j < n$.
Output: The orthogonal lattice $L_n^\perp$.
Create the matrix $K_n = (k_{i,j})$, $0 \leq i < n-d$ and $0 \leq j < n$, where the entries $k_{i,j}$ are initially set to 0 ;
for *i from* 0 *to* $n-d-1$ **do**
  for *j from* 0 *to* $d-1$ **do**
    $k_{i,j} \leftarrow -m_{j,i+d}$;
    $j \leftarrow j+1$;
  end
  $k_{i,i+d} \leftarrow 1$;
  $i \leftarrow i+1$;
end
The rows of K_n constitute the basis vectors of $L_n^\perp$.

Algorithm 2: **Computing the orthogonal lattice $L_n^\perp$**

The $(n-d) \times n$ matrix representing the orthogonal lattice $L_n^\perp$ (the lattice basis vectors form the rows of the matrix) will have the following shape:

$$K_n = \begin{pmatrix} -m_{0,d} & \ldots & -m_{d-1,d} & 1 & 0 & 0 & \ldots & 0 \\ -m_{0,d+1} & \ldots & -m_{d-1,d+1} & 0 & 1 & 0 & \ldots & 0 \\ \vdots & & \vdots & \vdots & & \ddots & & \\ -m_{0,n-1} & \ldots & -m_{d-1,n-1} & 0 & 0 & \ldots & 0 & 1 \end{pmatrix}$$

Lemma 1. *Algorithm 2 computes a basis of the orthogonal lattice $L_n^\perp$ with running time $\mathcal{O}((n-d)d)$.*

Proof. It is clear that Algorithm 2 runs in time $\mathcal{O}((n-d)d)$. It remains to prove that it actually computes a basis of the orthogonal lattice $L_n^\perp$. Since the matrix K_n has $(n-d)$ rows and the lattice $L_n^\perp$ has dimension $n-d$ according to Theorem 3, it suffices to prove that the rows of K_n form a generating family of $L_n^\perp$. Let $v = (v_0, \ldots, v_{n-1}) \in L_n^\perp$. Then by definition of the orthogonal lattice, $< v, u >= 0, \forall u \in L_n$, for instance $< v, u_i >= 0$ for all the lattice L basis vectors u_i (rows of the matrix M_n). It follows that $\sum_{0 \le j \le n-1} m_{i,j} v_j = 0$ and thus $v_i = -\sum_{d \le j \le n-1} m_{i,j} v_j$ for $0 \le i \le d-1$. Consequently,

$$\begin{aligned} v &= (v_0, \ldots, v_n) \\ &= (-\sum_{d \le j \le n-1} m_{0,j} v_j, \ldots, -\sum_{d \le j \le n-1} m_{d-1,j} v_j, v_d, \ldots, v_{n-1}) \\ &= \sum_{d \le j \le n-1} v_j (-m_{0,j}, \ldots, -m_{d-1,j}, 0, \ldots, 1, 0, \ldots, 0). \end{aligned}$$

v can then be written as a linear combination of the rows of K_n with coefficients v_j, $d \le j \le v_{n-1}$, which concludes the proof.

□

Finding the low weight polynomial multiples. This is the most expensive part of Algorithm 1 since it corresponds to the basis reduction of the orthogonal lattice $L_n^\perp$. The LLL reduction can be performed in $\mathcal{O}(n(n-d)^5)$[3]. This means that the higher the dimension, the more infeasible the attack gets.

Theorem 5. *Algorithm 1, in case the reduction applied is LLL, runs in $\mathcal{O}(n(n-d)^5)$ arithmetic operations, and computes $(n-d)$ multiples of the polynomial f of weight w_i: $w_i \le 2^{n-d-1} \lambda_i(L_n^\perp)^2, 1 \le i \le n-d$, where $\lambda_i(L_n^\perp)$ denote the successive minima of the lattice $L_n^\perp$, for instance $\lambda_1(L_n^\perp)$ is the shortest nonzero vector in the lattice which corresponds also to the lowest weight of f's multiples of degree at most $n-1$.*

Proof. We first show that Algorithm 1 computes multiples of f. Let $v = (v_0, \ldots, v_{n-1}) \in L_n^\perp$. Then $\sum_{j=0}^n v_j m_{i,j} = 0$, $0 \le i \le d-1$. It follows that $\sum_{j=0}^n v_j (X^j \bmod f) = 0$, thus, $\sum_{j=0}^n v_j (X^j \bmod f) \equiv 0 \bmod 2$ or equivalently $f | \sum_{j=0}^n (v_j \bmod 2) X^j$. To prove the running time as well as the bound on the weights of the resulting multiples, we just refer to the famous LLL paper [14] where the the authors prove the approximation factors of the reduced basis vectors with regard to the successive minima when the LLL reduction is applied.

□

Remark 1. In order to improve on the quality of the obtained basis, we could use, instead of the LLL reduction, Schnorr's reduction algorithm [24] or the recently improved algorithm [1]. We will obtain then approximation factors that are slightly sub-exponential , namely - $2^{\mathcal{O}((n-d)(\log\log(n-d))^2/\log(n-d))}$ and $2^{\mathcal{O}((n-d)\log\log(n-d)/\log(n-d))}$ respectively. Note that an exact solution of the lowest weight multiple (or even an approximation to within polynomial factors in the excess degree $(n-d)$), can be achieved in exponential running time.

[3] Actually LLL runs in $\mathcal{O}(d^5 n) B^3$, where d and n represent the lattice and vector space dimensions resp. and B is an upper bound on the coefficients'size of the inupt basis vectors. In our case these have values in $\{0, 1\}$, thus $B = 1$.

Remark 2. There exists also a heuristic that estimates the vectors lengths in a reduced basis output by the algorithm 1 by the product of the square root of the dimension $n-d$ and the $(n-d)$-th root of the lattice determinant. This gives us multiples of weights with approximation factors polynomial in $(n-d)$ to the actual minimal weight.
On the practical side, the LLL algorithm, despite its pessimistic theoretical bounds, achieves a basis with moderately short vectors. For instance, all the resulting vectors in the reduced basis of $L_n^{\perp}$ have entries in $\{0, \pm 1\}$ and if the dimension is small enough, then we find multiples with the lowest possible weight.

In order to relate the shortness of the obtained reduced basis vectors and the sparseness of the resulting polynomials, we make the following assumption which is actually an empirical result, run over several instances of the problem, that we could not prove theoretically

Assumption 1 *Algorithm 2 outputs a basis for the orthogonal lattice with vectors having entries in* $\{0, \pm 1\}$

In this way, the weight of the resulting polynomials will be nothing but the square of the basis vectors' ℓ_2-norm.

3.2 Previous work

The strategy used so far to solve this problem consists of first estimating the minimal weight w of multiples of the given polynomial f with degree at most $n-1$, then finding multiples of weight at most w. To estimate the minimal weight, one solves for w_d the following inequality; w is the smallest solution: $2^{-d}\binom{n-1}{w_d} \geq 1$. In fact, if multiples were random then one expects that the above inequality holds. It is worth mentioning that the number of such multiples could be approximated by $\mathcal{N}_M = 2^{-d}\binom{n-1}{w-1}$.

The techniques used to find sparse multiples of weight at most w are:

- **Exhaustive search.** When the bound n is just above d, an exhaustive search turns out to be faster. The cost of such an attack is $\mathcal{O}(\mathcal{P}oly(d) \cdot 2^{n-d-1})$.
- **Syndrome decoding.** We compute the matrix H whose columns are defined by $H_i = X^i \bmod f$, $1 \leq i \leq n-1$, then find a low weight word in the preimages of 1 of this matrix. The cost of this method is $\mathcal{O}(\mathcal{P}oly(n-1)(\frac{n-1}{d})^{w-1})\mathcal{N}_M$, where $\mathcal{P}oly$ is a polynomial of degree 2 or 3.
- **The birthday Paradox [16].** Set $w = q_1 + q_2 + 1$, $q_1 \leq q_2$, and build two lists; the first one contains all possible linear combinations of $X^i \bmod f$, $0 < i < n$ of weight q_1 whereas the second list contains all possible linear combinations of $X^i \bmod f$, $0 < i < n$ of weight q_2. Then look for pairs that sum to 1. Clearly, this method runs in $\mathcal{O}(n^{q_2})$ (if we implement the first list by an efficient hash-table), and uses $\mathcal{O}(n^{q_1})$ of memory. The usual time-memory trade-off is to use $q_1 = \lfloor \frac{w-1}{2} \rfloor$ and $q_2 = \lceil \frac{w-1}{2} \rceil$ in order to balance the cost of the two phases. Note that the running time depends on the parity of w since we do not have to compute anything if $q_1 = q_2$. There exist many improvements of this method, for example Chose et al. [5] use the *match-and-sort* alternative that consists of splitting the huge task of finding collisions among n^w combinations into smaller tasks: finding less restrictive collisions on smaller subsets, sort the results and then aggregate these intermediate results to solve the complete task. This leads to a considerable improvement of the space complexity, namely $\mathcal{O}(n^{\lceil w-1/4 \rceil})$. Didier and Laigle-Chapuy [7] consider a new approach that uses discrete logarithms instead of the direct representation of the involved polynomials. They achieve

a time/space complexity of $\mathcal{O}(n^{L\lfloor (w-1)/2 \rfloor})$, where L is the cost of computing a discrete logarithm in $\mathbb{F}_{2^d}$, and $\mathcal{O}(n^{\lfloor (w-2)/2 \rfloor})$ respectively.

– **Wagner's generalized birthday paradox.** When the bound n on the multiples'degree increases, then Wagner's generalized birthday paradox [29] becomes more efficient. In fact, if there exists $a \geq 2$ such that $\binom{n-1}{(w-1)/2^a} \geq 2^{d/(a+1)}$, then one can find a solution in $\mathcal{O}(2^a 2^{d/(a+1)})$. For instance, if $n \geq 2^{d/(1+\log_2(w-1))}$, using this method, one can find a multiple within almost linear time in n, namely, $\mathcal{O}((w-1)n)$.

We summarize the costs (time and complexity) of the different methods in the following table:

Method	Exhaustive Search	Syndrome Decoding	Birthday Paradox	Generalized BP	Our method
Time cost	$\mathcal{O}(\mathcal{P}oly(d) \cdot 2^{n-d-1})$	$\mathcal{O}(\mathcal{P}oly(n-1)(\frac{n-1}{d})^{w-1})\mathcal{N}_M$	$\mathcal{O}(n^{\lceil \frac{w-1}{2} \rceil})$	$\mathcal{O}((w-1)2^{d/(1+\log_2(w-1))})$	$\mathcal{O}((n-d)^5 n)$
Space cost	$\mathcal{O}(n)$	$\mathcal{O}(\mathcal{P}oly(n-1)(\frac{n-1}{d})^{w-1})$	$\mathcal{O}(n^{\lceil w-1/4 \rceil})$	$\mathcal{O}(2^{d/(1+\log_2(w-1))})$	$\mathcal{O}(nd)$

From the list above, we conclude that our algorithm achieves a better cost when the weight w increases, say gets bigger than 12. Also, In the case where the excess degree $n-d$ is small (but not too small such that an exhaustive search is feasible); one concrete example is when $w = 3$ and and $n > (n-d)^5$ or when $w = 5$ and $n^2 > (n-d)^5$.

3.3 Experiments

To validate our method, we tested it on some known polynomials, using the NTL library [26] developed by Victor Shoup on a 2.2-GHz Athlon processor with 2 GB of RAM. More precisely, we used for the lattice basis reduction (step 3 in Algorithm 1) two implementations of floating LLL (and its variants). In fact, the (original) LLL algorithm operates on rationals in order to compute the Gram-Schmidt orthogonalization coefficients. In big dimensions, the size of these latter items increases and makes the algorithm impractical, thus one is tempted to approximate the mentioned coefficients using a floating point representation. We basically use the NTL LLL_FP algorithm which represents an improvement of the Schnorr-Euchner version [25] that uses a double precision. In order to improve on the quality of the reduction, we also make use of a floating point implementation of the Block Korkin-Zolotarev basis reduction (in double precision as well), namely the BKZ_FP algorithm. This is slower but yields a higher-quality basis, i.e., one with shorter vectors. It basically generalizes the LLL reduction condition from blocks of size 2 to blocks of larger size. BKZ_FP is an implementation of the Schnorr-Euchner algorithm [25]. Finally, it is worth noting that the best fully proved floating point arithmetic LLL variant is due to Nguyen and Stehlé [20]. The so-called L^2 algorithm which runs in time $\mathcal{O}(d^4 n \log B(d+\log B))$, where d, n and B refer to the lattice dimension, the vector space dimension and an upper bound on the lattice basis vectors' norm.
We got the following results (n refers to the strict upper bound on the polynomial, w_e and w_f refer to the estimated minimal weight and the smallest weight found resp. , finally M and t denote the number of multiples found of degree at most w_f and the corresponding time (in seconds) resp.):

Experiment 1. $f_1 = 1+X^2+X^4+X^5+X^6+X^8+X^9+X^{10}+X^{11}+X^{13}+X^{14}+X^{15}+X^{17}$.

1. The Lattice method using the floating variant of LLL (LLL_FP) from the NTL library with the default parameters:

$n-1$	17	20	20	20	21	22	24	30	44	94	513
w_e	13	12	11	10	9	8	7	6	5	4	3
w_f	13	8	8	8	8	8	8	5	5	5	4
M	1	1	1	1	2	2	2	1	1	1	2
t	0	0	0	0	0	0	0	0	0	0.004	0.696

2. The Lattice method using the floating variant of LLL (BKZ_FP) from the NTL library with 30 for the block size and 15 for pruning:

$n-1$	17	20	20	20	21	22	24	30	44	94	513
w_f	13	8	8	8	8	8	8	5	5	5	3
M	1	1	1	1	2	2	2	1	1	3	1
t	0	0	0	0	0	0	0	0	0	0.06	25.1

3. The Time-Memory Trade-Off (TMTO) using the C++ standard library (STL) hash function:

$n-1$	17	20	20	20	21	22	24	30	44	94	513
w_f	13	8	8	8	8	8	8	5	5	5	3
M	1	1	1	1	2	2	4	1	1	27	1
t	0	0.38	0.38	0.38	0.5	0.75	1.62	0.02	0.08	2.77	0.03

We provide in Appendix B the remaining experiments, where the TMTO method has been running for several days without any results.

Remarks. In order to evaluate experimentally the time estimate of our method, we utilized the linear regression tool to express the relationship between the logarithm of the time estimate ($\ln(t)$) and the logarithm of the bound on the multiples ($\ln(n)$). We got the following for the first experiment:

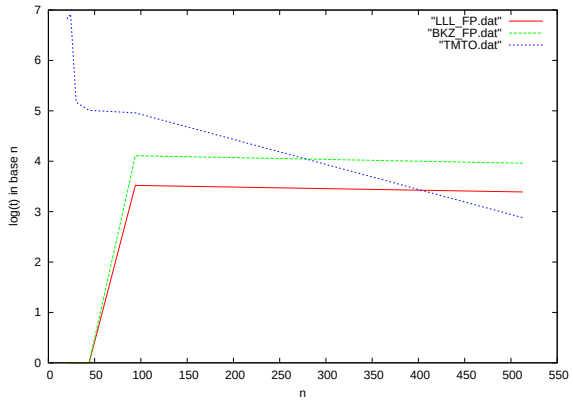
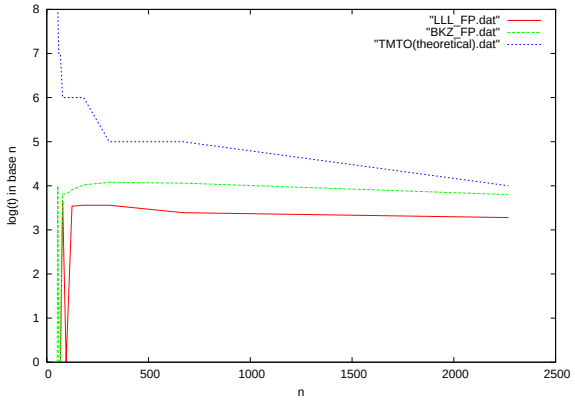


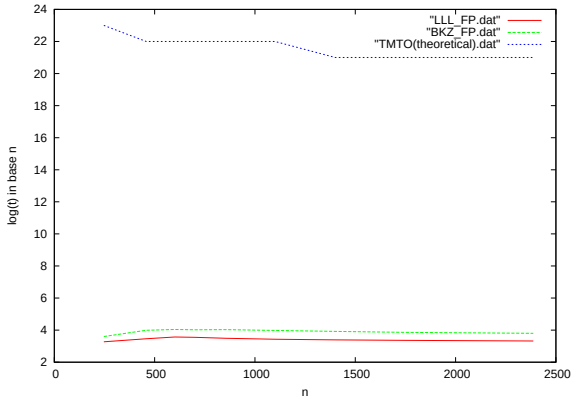
Fig. 1. Polynomial f_1

We first notice that for the TMTO method, the coefficient $\alpha = \log_n(t)$ is not always equal to $\lceil \frac{w-1}{2} \rceil$ as it should be. This is explained by the fact that the time estimate for this method, that is, $\mathcal{O}(n^{\lceil (w-1)2 \rceil})$, is only the best case complexity. In fact, the search in a hash table can be performed in constant time in the best case and linear time in the worst. It might be wiser then to use a more efficient hash table than the one provided by the standard library (STL) of C++.

We can also relate this to the unfruitful execution of the algorithm when the heuristic predicts a weight that doesn't exist.
Next, we note that the coefficient α of LLL_FP and BKZ_FP is constant and about 3.5 and 4 respectively. This explains why the TMTO method looses the lead as soon as the weight w gets greater than 8.
We did the same for the polynomials f_2 and f_3. Since we were not able to run the TMTO method on these instances, we defined the corresponding coefficient as $\alpha = \lceil \frac{w-1}{2} \rceil$, which corresponds to the best possible time estimate one could obtain. It is clear from the graphs that the lattice method is better since the weights that come into play are pretty big.



(a) **Polynomial f_2**



(b) **Polynomial f_3**

From the results above, we conclude that our algorithm wins in cases that are intractable for the previously known birthday-based methods, namely when the target weight is big and also when the excess degree is small.

4 Estimation of the Minimal Weight

As mentioned earlier in this document, the methods used in the literature to find low weight polynomial multiples, namely *Time-Memory Trade-Off* and *Syndrome decoding*, estimate first the expected minimal weight w_d, then find multiples of weight at most w_d. The way used to estimate the minimal weight is to solve for w the following inequality; w_d is the smallest solution to $2^{-d}\binom{n-1}{w} \geq 1$

This estimation is quite reasonable and works well in practice. However, since it is independent of the polynomial in question, there exist polynomials for which the estimation leads to a smaller or greater minimal weight. The problem is posed when the estimation outputs a weight that is much smaller that the actual minimal weight since this would correspond to running unproductively the algorithms to find sparse multiples until we hit the "correct" weight. It is worth noting that from a stream ciphers designer's perspective, these polynomials are regarded as secure as they are resistant to this strategy. The Bluetooth polynomial is an illustration of such polynomials.
In this section, we supply these methods by providing lower and upper bounds for the estimated minimal weight. In this way, the techniques will proceed to find sparse multiples only if the estimated weight is in the correct range. This result can be interpreted from stream ciphers designers as follows: given a polynomial (feedback polynomial of a constituent input LFSR), they insure that an adversary cannot find a multiple with weight below a certain bound given

access to a certain amount of keystream.
The bounds follow directly from Theorem 1. Moreover, according to Theorem 2, we have $\det(L_n^{\perp}) = \det(\bar{L}) = \det(L)$ since the coordinates of L_n basis vectors are in $\{0,1\}$. This results in the following:

$$\lceil \min_j \|K_j^*\|^2 \rceil \leq w_d \leq \lfloor \gamma_{n-d} \det(L)^{2/(n-d)} \rfloor,$$

where $(K_{i,j})^*$ represents the Gram-Schmidt data for the matrix K_n. We obtained the following bounds for polynomial f_1 (lb and ub refer to the lower and upper bounds on the minimal weight respectively whereas w_e denotes the estimated weight):

$n-1$	20	21	22	24	30	44	94	513
lb	5	5	5	5	3	2	2	2
w_e	10	9	8	7	6	5	4	3
ub	10	10	11	12	6	9	15	60

The remaining experiments are given in Appendix C.

In our computations, we used the following, known so far, values for the Hermite's constant:

$n-d$	2	3	4	5	6	7	8	24
γ_{n-d}	$2/\sqrt{3}$	$2^{1/3}$	$\sqrt{2}$	$8^{1/5}$	$(64/3)^{1/6}$	$64^{1/7}$	2	4

and for the other dimensions, we used these asymptotic bounds, which are known to be the best:

$$\frac{n-d}{2\pi e} + \frac{\log(\pi(n-d))}{2\pi e} + o(1) \leq \gamma_{n-d} \leq \frac{1.744(n-d)}{2\pi e}(1+o(1)).$$

First, we notice that the heuristic method works well in practice, since all the estimated weights fall in the provided range. Next, we remark that as soon as the dimension increases, the upper bounds become trivial and could be replaced simply by the weight of the given polynomial. It is clear then that exact values or even tighter bounds for the Hermite constant would lead to tighter upper bounds for the minimal weight.

5 Sparse Binary Multiples of an integer

The method applied above to find sparse multiples for a given polynomial applies also for searching sparse binary multiples of a given integer.
In fact, we define the following problem: we have integers n and g with g dividing $b^n - 1$, and consider the set S of all multiples ag with $0 < a < (b^n - 1)/g$. We want to know the smallest Hamming weight of the b-ary representations in S.
In the case $b = 2$, the same analysis is almost applicable. In fact, we are searching sparse multiples of the given number g that are smaller $2^n - 1$, i.e., multiples whose bitsizes are smaller than the bound [4] n.

The algorithm is then described as follows:

[4] This bound represents the order of the 2 in the multiplicative group $\mathbb{Z}_g$. Similarly, we searched sparse multiples, of a given polynomial f, of degree less than n, where n is smaller than the order of the polynomial f, otherwise the multiple $X^{\mathrm{ord}(f)} + 1$ would have been a solution.

Input: a number g of size d diving $2^n - 1$.
Output: low weight multiples of g smaller than $2^n - 1$

1. Compute the lattice n-dimensional lattice K_n defined by
   $K_n = \{(x_0, \ldots, x_{n-1}) \in \mathbb{Z}^n : \sum_{i=0}^{n-1} x_i 2^i = 0 \bmod g\}$;
2. Reduce it . The basis vectors with entries in $\pm\{0, 1\}$ constitute the low weight multiples.

Algorithm 3: **Computing sparse multiples of a given number**

In this situation, the computation of the orthogonal lattice can be also achieved in linear time in n using the following remark. An element of the lattice K_n is either a multiple of g (the coordinates of the element constitute its 2-ary representation), thus it belongs to the one dimensional lattice $g\mathbb{Z}$ and therefore generated by the vector $v = (g, 0, \ldots, 0)$, or it belongs to the sub-lattice $L^{\perp} = (1, \ldots, 2^{n-1})^{\perp}$, which is generated by vectors $v_1 = (-2, 1, 0, \ldots, 0), v_2 = (-4, 0, 1, 0, \ldots, 0), \ldots, v_{n-1} = (-2^{n-1}, 0, \ldots, 0, 1)$. Hence, $(v_1, \ldots, v_{n-1}, v)$ is a basis of K_n.
Note that the sparse multiples of g are only those which have entries in $\pm\{0, 1\}$. We can get rid of the parasite vectors, namely those with entries in $\{\pm 1, 0\}$ by considering the lattice $K'_n = 2K_n + \mathbb{Z}(1, \ldots, 1)$. We are then guaranteed that the desired vectors will appear in any reduced basis of the new lattice.
Full details about this problem and its applications to error-correcting codes will appear elsewhere.

We run again our experiments with NTL and got the following:

(g, n)	(268501,100)	(3173389601,200)	(10567201,300)	(82471201,400)
w_f	3	3	2	5
M	27	18	132	2
t	0.06	0.3	1.02	1.66

6 General Thoughts and Prospectives

An interesting question is to study the special form of the lattice $L_n^{\perp}$ or equivalently of the matrix K_n in order to reduce the cost of the reduction, or improve on the gotten results. In fact, the matrix in question is sparse and one is tempted to use a more compact representation or at least a representation that makes easy for the basis reduction algorithms, namely LLL and its variants, the search for short vectors. For instance, we noticed that changing the shape of the matrix such that the first $(n - d)$ columns form the unit matrix $I_{n-d} \in \mathbb{R}^{n-d}$ - in the original lattice, the last $(n - d)$ columns formed the unit matrix - leads to different results but still not spectacular.
Besides, according to Algorithm 1, short vectors in the lattice $L_n^{\perp}$ lead to sparse multiples of the polynomial f. So we managed to relate the hardness of the low weight polynomial multiple problem to the hardness of the shortest vector problem in $L_n^{\perp}$. We can also relate the closest vector problem to our problem, in fact, a lattice point in $L_n^{\perp}$ close to the constant polynomial 1 will lead to a low weight multiple of nonzero constant term. This suggests to study the hardness of the shortest/closest vector problems of this special instance of lattices (lattice of the form K_n) in order to better estimate the hardness of the low weight polynomial multiple problem. We believe the taxonomy: sparse polynomial multiple problem - shortest/closest vector problem of lattices with form K_n - syndrome decoding, deserves further attention. In fact, this would provide us either with very efficient tools to solve the problem and hence lead to new improvements in stream ciphers cryptanalysis and fast finite field arithmetic, or with confidence on the hardness of the problem (if we manage to exhibit a reduction from syndrome decoding or SVP/CVP to it), since the other two problems are known to be NP-complete. The last point is motivated by

the recent proposal of a cryptosystem whose security relies on the problem [9].
Furthermore, the reduced basis of K_n contains short vectors or equivalently sparse multiples that do not have necessarily nonzero constant term. This is due to the following fact: if g is a sparse multiple with nonzero constant term, then there is no restriction on the basis to contain the multiples $X^i g$ granted that $\deg(g)+i < n$. This leads to redundancies in the basis. It would then be desirable if one filters out extraneous polynomials in order to allow more "interesting" multiples to appear in the basis. One way to achieve this is to compute points in the lattice $L_n^{\perp}$ that are close to the constant polynomial 1. The cost of such a technique will be about the same since we will use the famous *embedding technique*, which consists of reducing the $(n+1-d)$-dimensional lattice $L'_n \subseteq \mathbb{Z}^{n+1}$ given by the basis $\mathcal{K}' = (k'_1, \ldots, k'_{n+1-d})$, where $k'_i = (k_i, 0)$, $1 \leq i \leq n-d$, and $k_{n+1-d} = (1, 0, \ldots, 0, 1)$. Experiments curried out improved slightly the results, for instance, we got a further nonzero multiple of f_1 of degree at most 94. The weak impact of this strategy lies in the small CVP-gap, i.e., the ratio between the shortest vector of L'_n and the distance of the constant polynomial 1 to it. In fact, the embedding technique requires a large gap in order to give accurate results. It would be interesting to dig further in this direction, for example solve directly the CVP instance instead of reducing it to a SVP instance. Experiments are in progress and will appear in the full version of the paper.
Finally, one is tempted to extend the method into finding low weight multiples of polynomials over $\mathbb{F}_p$, where $p > 2$ is a prime number (for the corresponding number problem, consider the case $b > 2$). However, the naive approach would not work since the correspondence short vector / sparse multiple won't hold anymore.

7 Summary

We have proposed a new algorithm to find low weight multiples for a given polynomial of degree at most n using lattice basis reduction. The method has a theoretical time estimate of $\mathcal{O}((n-d)^5 n)$, in case LLL is the reduction algorithm used, and an experimental one about $\mathcal{O}(n^4)$. It takes then the lead as soon as the expected minimal weight w gets bigger that 8. In fact, the best known methods, that are the birthday-based ones, have a best-case time estimate about $\mathcal{O}(n^{\lceil (w-1)/2 \rceil})$. Such a situation occurs when the bound on the multiple, which denotes the available keystream, is small. We also gave bounds for the minimal weight of such polynomials using lattices. This helps to check whether the estimated weight, furnished by the state-of-the art methods is in the correct range. Also, it provides a quality criterion, from a stream ciphers designer 's stand, on the used polynomial. In fact, designers insure that the adversary cannot find multiples with weight below a certain bound, given access to a certain amount of keystream. Furthermore, we introduced the corresponding number problem and applied almost the same technique to find sparse binary multiples of a given integer. Finally, we confirmed our analysis by implementing the method using NTL; our method is applicable for relatively high dimensions (up to 2000), using the floating variants of LLL, and has proved very efficient on instances that are intractible for the the standard methods.

References

1. R. Ajtai, M. Kumar and D. Sivakumar, *A sieve algorithm for the shortest lattice vector problem*, 33rd ACM Symp. on Theory of Computing, 2001, pp. 601–610.
2. S. Brent and P. Zimmermann, *Algorithms for finding almost irreducible and almost primitive trinomials*, Lectures in Honour of the Sixtieth Birthday of Hugh Cowie Williams (2003).
3. S. Brent, R. Larvala and P. Zimmermann, *Fast algorithm for testing reducibility of trinomials mod 2 and some new primitive trinomials of degree 3021377*, Math. Comput **72** (2003), 1443–1452.
4. A. Canteaut and M. Trabbia, *Improved Fast Correlation Attacks Using Parity-Check Equations of Weight 4 and 5*, Advances in Cryptology - EUROCRYPT 2000 (B. Preneel, ed.), Lect. Notes Comput. Sci., vol. 1807, Springer, 2000, pp. 573,588.
5. A. Chose, P. Joux and M. Mitton, *Fast correlation attacks: an algorithmic point of view*, Advances in Cryptology - EUROCRYPT 2002 (L. R. Knudsen, ed.), Lect. Notes Comput. Sci., vol. 2332, Springer, 2002, pp. 209,221.
6. H. Cohen, *A course in computational algebraic number theory.*, Graduate Texts in Mathematics 138., Springer-Verlag, 1996.
7. J. Didier and Y. Laigle-Chapuy, *Finding low-weight polynomial multiples using discrete logarithm*, IEEE INTERNATIONAL SYMPOSIUM ON INFORMATION THEORY ISIT'07 (Nice, France), June 2007.
8. P. Ekdahl and T Johansson, *Distinguishing Attacks on SOBER-t16 and t32*, Advances in Cryptology - FSE 2002 (Joan Daemen and Vincent Rijmen, eds.), Lecture Notes in Computer Science, vol. 2365, Springer, 2002, pp. 210,224.
9. M. Finiasz and S. Vaudenay, *When stream cipher analysis meets public-key cryptography*, SAC 2006 (Eli Biham and Amr M. Youssef, eds.), Lect. Notes Comput. Sci., vol. 4356, Springer, 2006, pp. 266–284.
10. R.G. Gallager, *Low-density parity-check codes.*, IRE Trans. Inform. Theory **IT-8** (1962), 21–28.
11. T. Johansson and F. Jönsson, *Fast correlation attacks based on turbo code techniques codes*, Advances in Cryptology - CRYPTO 1999 (M. J. Wiener, ed.), Lect. Notes Comput. Sci., vol. 1666, Springer, 1999, pp. 181,197.
12. ———, *Improved fast correlation attack on stream ciphers via convolutional codes*, Advances in Cryptology - EUROCRYPT 1999 (J. Stern, ed.), Lect. Notes Comput. Sci., vol. 1592, Springer, 1999, pp. 347,362.
13. ———, *Fast correlation attacks through reconstruction of linear polynomials*, Advances in Cryptology - CRYPTO 2000 (M. Bellare, ed.), Lect. Notes Comput. Sci., vol. 1880, Springer, 2000, pp. 300,315.
14. H. Lenstra, A. Lenstra and L. Lovasz, *Factoring polynomials with rational coefficients*, Math. ann **261** (1982), no. 4, 515–534.
15. Y. Lu and S. Vaudenay, *Faster correlation attack on Bluetooth keystream generator E0*, Advances in Cryptology - CRYPTO 2004 (M. K. Franklin, ed.), Lect. Notes Comput. Sci., vol. 3152, Springer, 2004, pp. 407,425.
16. W. Meier and O. Staffelbach, *Fast correlation attacks on certain stream ciphers*, J. Cryptology (1989), 159–176.
17. A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography.*, Discrete Mathematics and its Applications., CRC Press, Boca Raton, FL, 1997.
18. D. Micciancio and S. Goldwasser, *Complexity of lattice problems - A cryptographic perspective.*, Kluwer Academic Publishers, 2002.
19. P Nguyen, *La géométrie des nombres en cryptologie*, Ph.D. thesis, Laboratoire d'Informatique de l'École Normale Supérieure, France, November 1999.
20. P. Nguyen and D. Stehlé, *Floating-Point LLL Revisited*, Advances in Cryptology - EUROCRYPT 2005 (R. Cramer, ed.), Lect. Notes Comput. Sci., vol. 3494, Springer, 2005, pp. 215–233.
21. P. Nguyen and J. Stern, *Merkle-Hellman Revisited: A cryptanalysis of the Qu-Vanstone cryptosystem based on group factorizations* , Advances in Cryptology - CRYPTO 1997 (B. S. Kaliski Jr., ed.), Lect. Notes Comput. Sci., vol. 1294, Springer, 1997, pp. 198,212.
22. ———, *The Béguin-Quisquater server-aided RSA protocol from Crypto '95 is not secure* , Advances in Cryptology - ASIACRYPT 1998 (K. Ohta and D. Pei, eds.), Lect. Notes Comput. Sci., vol. 1514, Springer, 1998, pp. 372–379.
23. ———, *Cryptanalysis of a fast public key cryptosystem presented at SAC '97*, SAC 1998 (Stafford E. Tavares and Henk Meijer, eds.), Lecture Notes in Computer Science, vol. 1556, Springer, 1999, pp. 213–218.
24. C.-P. Schnorr, *A hierarchy of polynomial time lattice basis reduction algorithms*, Theoretical Computer Science **53** (1987), no. 2-3, 201–224.
25. C. P. Schnorr and M. Euchner, *Lattice basis reduction: improved practical algorithms and solving subset sum problems*, FCT, Lect. Notes Comput. Sci., vol. 591, Springer, 1991, pp. 68–85.
26. V. Shoup, *NTL: a library for doing number theory.*, Available online at `http://www.shoup.net/ntl/`.
27. T. Siegenthaler, *Decrypting a class of stream ciphers using ciphertext only.*, IEEE Trans.Computers **C-34** (1985), no. 1, 81–84.
28. J. von zur Gathen and M. Nöcker, *Polynomial and normal bases for finite fields*, J.Cryptology **18** (2005), no. 4, 337–355.
29. D. Wagner, *A Generalized Birthday Problem.*, Advances in Cryptology - CRYPTO 2002 (M. Yung, ed.), Lect. Notes Comput. Sci., vol. 2442, Springer, 2002, pp. 288–304.

A Proof of Theorem 2

Proof. First, we note that if $(b_1, \ldots, b_d)$ with $b_i = (b_i^1, \ldots, b_i^n)$ is a basis for L, then $(\frac{1}{g_1}b_1, \ldots, \frac{1}{g_d}b_d)$ is a basis for $\bar{L}$, where $g_i = \gcd(b_i^1, \ldots, b_i^n)$, $1 \leq i \leq d$. In fact, Let $x \in \bar{L}$, then there exist $x_1, \ldots, x_d \in \mathbb{R}$ such that $x = \sum_{1 \leq i \leq d} x_i b_i = \sum_{1 \leq i \leq d} x_i g_i \frac{1}{g_i} b_i$ and $x \in \mathbb{Z}^n$. Since $\frac{1}{g_i}b_i \in \mathbb{Z}^n$, it follows that $x \in \mathbb{Z}^n$ if and only if $x_i g_i \in Z^n$, $\forall 1 \leq i \leq d$. So we managed to write x as a linear combination, with integer coefficient, of the $\frac{1}{g_i}b_i$'s. Since these latter vectors are linearly independent, $(\frac{1}{g_1}b_1, \ldots, \frac{1}{g_d}b_d)$ is then a basis of $\bar{L}$.
Next, we apply the definition of the determinant of a lattice and get the announced result.

□

B Experiments

B.1 Experiment 2.

$f_2 = 1 + X + X^3 + X^5 + X^9 + X^{11} + X^{12} + X^{17} + X^{19} + X^{21} + X^{125} + X^{27} + X^{29} + X^{32} + X^{33} + X^{38} + X^{40}$.

1. The lattice method using LLL_FP:

$n-1$	40	49	51	54	59	67	78	95	124	181	307	669	2268
w_e	17	16	15	14	13	12	11	10	9	8	7	6	5
w_f	17	17	16	15	15	13	13	13	13	13	11	9	8
M	1	1	1	1	1	1	1	1	2	2	1	1	1
t	0	0	0	0	0	0	0.004	0	0.012	0.05	0.34	1.78	49.75

2. The lattice method using BKZ_FP:

$n-1$	40	49	51	54	59	67	78	95	124	181	307	669	2268
w_f	17	17	16	15	15	13	13	13	12	12	11	10	8
M	1	1	1	1	1	1	1	1	1	2	1	1	1
t	0	0	0	0.004	0	0	0.008	0.016	0.07	0.56	6.38	142.8	2506.77

3. The TMTO method:

$n-1$	40	49	51	54	59	67	78	95	124	181	307	669	2268
t	0	-	-	-	-	-	-	-	-	-	-	-	-

B.2 Experiment 3.

The Bluetooth polynomial (multiple of the four constituent LFSRs feedback polynomials) ; $f_3 = f_3^1 \cdot f_3^2 \cdot f_3^3 \cdot f_3^4$ where:

$$f_3^1(x) = x^{25} + x^{20} + x^{12} + x^8 + 1;$$

$$f_3^2(x) = x^{31} + x^{24} + x^{16} + x^{12} + 1;$$

$$f_3^3(x) = x^{33} + x^{28} + x^{24} + x^4 + 1;$$

$$f_3^4(x) = x^{39} + x^{36} + x^{28} + x^4 + 1;$$

1. The lattice method using LLL_FP

$n-1$	128	247	458	600	700	855	1100	1400	1749	2387
w_e	49	31	24	23	22	20	19	18	17	16
w_f	49	47	47	47	47	47	47	41	41	41
M	1	1	1	1	1	1	1	1	1	1
t	0	0.03	0.74	3.8	5.9	7.9	12.4	20.7	35.11	77.03

2. The lattice method using BKZ_FP

$n-1$	128	247	458	600	700	855	1100	1400	1749	2387
w_f	49	47	44	44	44	44	44	44	44	44
M	1	1	1	1	1	1	1	1	1	1
t	0	0.184	18.32	75.97	123.2	305.3	600	970.82	1520.62	3138

3. The lattice method using BKZ_FP

$n-1$	128	247	458	600	700	855	1100	1400	1749	2387
$n-1$	0	-	-	-	-	-	-	-	-	-

C Estimation of the Minimal Weight

– The polynomial f_2:

$n-1$	49	51	54	59	67	78	95	124	181	307	669	2268
lb	11	10	8	6	4	4	3	2	2	2	2	2
w_e	16	15	14	13	12	11	10	9	8	7	6	5
ub	12	15	17	20	24	28	32	35	39	51	89	255

– The Bluetooth polynomial f_3:

$n-1$	247	458	522	604	855	1053	1334	1749	2387
lb	7	2	2	2	2	2	2	2	2
w_e	31	24	23	22	20	19	18	17	16
ub	218	174	171	171	183	199	225	266	330