Finding Low Weight Polynomial Multiples Using Lattices

Laila El Aimani and Joachim von zur Gathen

b-it, Dahlmannstr. 2, Universität Bonn, 53113 Bonn, Germany

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 $\mathcal{O}(n^6)$ and uses $\mathcal{O}(nd)$ of memory. This improves the space needed and gives a better theoretical time estimate when $w \geq 12$. Such a situation is plausible when the bound n, which represents the available keystream, is small. 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, i.e. a multiple of low Hamming weight, of a polynomial over $\mathbb{F}_2$ is believed to be a difficult 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 has not been investigated to the best of the authors' knowledge.

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:

$$M_f(n, w) = \{g \in \mathbb{F}_2[X] \colon f | g, \deg(g) < n, \text{weight}(g) \leq w_0\},$$

where w_0 is the least possible weight: $w_0 = \min\{w \colon M_f(n, w) \neq \emptyset\}$. 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.

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 [14] 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 [24] and later improved by Meier and Staffelbach [13]. Since then, a series of proposals sprang up, either very general or adapted to a specific scheme, to name but a few [9, 8, 10, 3, 4]. 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 $p = \mathrm{Prob}(s^i_k = s_k)$ 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 [7] 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 for 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 $\mathrm{Prob}(b_i = 0) = 1/2 + \gamma$, $\gamma > 0$[1], 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 [12] on E0, and [6] on SOBER-t16 and SOBER-t32.

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

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 [25], 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. They actually describe efficient algorithms for finding trinomials with large irreducible (and possibly primitive) factors and give examples of such trinomials.

Previous work

Most strategies used so far to solve this problem consist in 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 $\mathcal{O}(\mathcal{P}oly(d) \cdot 2^{n-d-1})$.
- **Finding Minimum Weight Words in a Linear Code.** The low weight polynomial multiple problem can be also solved by techniques that find a minimum weight word in a linear code. In fact, polynomial multiples form a linear code of length n, where n is the bound on the multiple's degree, and a low weight multiple corresponds to a minimum weight word in this code. There are known algorithms for performing this task, we note as a reference the algorithm by Canteaut and Chabaud [**?**], which runs empirically in $2^{ncH_2(1+R_0)+d}$, where $c = 0.12$, $d = 10$, $R_0 = 3.125\ 10^{-2}$ and H_2 corresponds to the entropy function. Besides, the algorithm uses approximately n^2 of memory. Note that this method, as the exhaustive search, does not estimate the minimal weight of the multiples.
- **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 [13].** 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. [4] 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 [5] 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 [26] 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 space) of the different methods in the following table:

Method	Exhaustive Search	Syndrome Decoding	Min Weight Words	Birthday Paradox	Generalized BP
Time cost	$\mathcal{O}(\mathcal{P}oly(d)\cdot 2^{n-d-1})$	$\mathcal{O}(\mathcal{P}oly(n-1)(\frac{n-1}{d})^{w-1})\mathcal{N}_M$	$\mathcal{O}(2^{ncH_2(1+R_0)+d})$	$\mathcal{O}(n^{\lceil\frac{w-1}{2}\rceil})$	$\mathcal{O}((w-1)2^{d/(1+\log_2(w-1))})$
Space cost	$\mathcal{O}(n)$	$\mathcal{O}(\mathcal{P}oly(n-1)(\frac{n-1}{d})^{w-1})$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^{\lfloor w-1/4\rfloor})$	$\mathcal{O}(2^{d/(1+\log_2(w-1))})$

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 a full dimensional lattice in $\mathbb{Z}^n$. Hence, it runs theoretically in $\mathcal{O}(n^6)$ (the entries of the matrix representing the lattice are in in $\{0, \pm 1, 2\}$) in case the LLL reduction is applied. This constitutes the first polynomial time algorithm for approximatively solving this problem. In fact, although the lattice basis reduction algorithms give only an approximation to the solution which is exponential in the lattice dimension, they are empirically known to perform better and lead to almost exact solutions when this dimension is sufficiently small. Therefore, our algorithm only supplements the known methods in specific ranges. For instance, it gives a better time estimate compared to the Time Memory Trade-Off (birthday technique) which runs in $\mathcal{O}(n^{\lceil(w-1)/2\rceil})$ in case w is big, say bigger than 12.

The rest of the paper is organized as follows; first, we give some preliminaries about lattices. 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 TMTO (Time Memory Trade-Off) technique in order to confirm our analysis. 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 [15] 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 than 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 $\sqrt{\Delta(L)}$.

Remark 1. In the rest of the document, if M is a matrix with rows $b_1, \ldots, b_d$, then we denote by $\mathcal{L}(M)$, the lattice generated by the vectors b_i's, $1 \leq i \leq d$. Similarly, if L is a lattice given by a generating family $(b_1, \ldots, b_d)$, then the matrix having for rows the vectors b_i's, $1 \leq i \leq d$, is denoted $\mathcal{B}(L)$.

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.*

We get now to the *orthogonal lattice*, a notion which was first introduced in a cryptanalytic context by Nguyen and Stern in 1997 [18]. It has proved very important and was used to attack many public key cryptosystems [18–20].

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 $\mathrm{span}(L)$.

Moreover, we have the following result [16, Chapter 2/Lemma 2.7]

Theorem 1. *If L is a lattice in $\mathbb{Z}^n$, then $\dim(L) + \dim(L^\perp) = n$.*

□

Finally, computing the orthogonal lattice amounts to determining the kernel of a matrix (as a $\mathbb{Z}$-module).

Theorem 2. *Given a basis of a d-dimensional lattice L in $\mathbb{Z}^n$, one can compute a basis of the orthogonal lattice $L^\perp$ in $\mathcal{O}((n-d)^2 d^3 |A|^2)$, where A is a bound on the bit size of the lattice basis entries.*

□

3 Finding Low Weight Polynomial Multiples Using Lattices

Let f be a polynomial of degree d over $\mathbb{Z}_2$ and let $n > d$ be a given bound. The task is to find a low weight multiple of f over $\mathbb{Z}_2$ with degree strictly less than n.

The idea underlying our approach is simple and based on the remark that such a polynomial multiple is a low weight linear combination with coefficients in $\{0, 1\}$ of the monomials x^i, $0 \leq i < n$, that evaluates to zero modulo f. Moreover, the polynomial multiples over $\mathbb{Z}_2$, with degrees at most $n-1$, form a lattice L_n in $\mathbb{Z}^n$. In fact, they are (a polynomial is given by its coefficients vector) in $\mathbb{Z}^n$, and the subtraction (or addition) of two multiples over $\mathbb{Z}_2$ is again a multiple over $\mathbb{Z}_2$. Therefore, searching a low weight polynomial multiple of f corresponds to searching a short vector in the lattice L_n. A high level description of the algorithm is depicted below.

Input : a polynomial f of degree d, and a bound $n > d$.

Output : multiples of f of degree less than n which are hopefully sparse.

1. Compute a basis of the lattice L_n ;
2. Reduce it using an appropriate lattice basis reduction algorithm ;
3. The resulting basis vectors constitute the desired polynomial multiples. For instance, if $v = (v_0, \ldots, v_{n-1})$ is a short basis vector, then $m = \sum_{1 \leq i < n} (v_i \bmod 2) x^i$ is a sparse multiple of f ;

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

In the sequel, we analyze the details of the above algorithm.

3.1 The lattice L_n

We define the set L_n of multiples of f over $\mathbb{Z}_2$ as follows:

$$L_n = \{g \in \mathbb{Z}_2[x] : g = 0 \bmod f, \deg(g) < n\}$$

Lemma 1. *L_n is a full-dimensional lattice in $\mathbb{Z}^n$.*

Proof. L_n is clearly a lattice since it is a subgroup of $\mathbb{Z}^n$.
Moreover, every polynomial, of degree at most $n-1$, with even coefficients is a multiple of f over $\mathbb{Z}_2$. Therefore L_n contains a family of n linearly independent vectors v_i, $1 \leq i \leq n$:

$$v_i(j) = \begin{cases} 2 \text{ if } j = i, \\ 0 \text{ otherwise.} \end{cases}$$

where $v_i(j)$ denotes the j-th entry of the n-length vector v_i. It is easy to see that the vectors v_i, $1 \leq i \leq n$, correspond to the polynomials 2, $2x$,...,$2x^{n-1}$. We conclude that L_n has dimension n.
□

Clearly the family of the above vectors v_i is not a basis of the lattice L_n since it does not span it. In fact, a multiple of f over $\mathbb{Z}_2$ is the addition of a multiple of f over $\mathbb{Z}$ and a polynomial of degree at most $n-1$ with even coefficients, i.e., a linear combination of the vectors v_i. Hence, the vectors v_i together with a basis of the multiples of f over $\mathbb{Z}$ constitute a generating family of the lattice L_n.

Computing a basis of the multiples of f over $\mathbb{Z}$. Let K_n be the set of multiple of f, over $\mathbb{Z}$ with degree at most $n-1$:

$$K_n = \{g \in \mathbb{Z}[x] \colon g = 0 \bmod f, \deg(g) \leq n-1\}$$

K_n is clearly a subgroup and therefore a lattice in $\mathbb{Z}^n$. We can actually view it as the orthogonal lattice of a d-dimensional lattice in $\mathbb{Z}^n$.

Let M_n be the $d \times n$ matrix whose columns are the coefficients of $h_i = x^i \bmod f$ for all $0 \leq i < n$. Let further $\mathcal{L}(M_n)$ denote the lattice, in $\mathbb{Z}^n$, generated by the rows of the matrix M_n. This lattice has dimension d since the first d components of its generators form a unit matrix, and thus the generators are linearly independent.

One can easily see that the orthogonal lattice $\mathcal{L}(M_n)^\perp$ is nothing but the lattice K_n. Hence, according to Theorem 1, K_n has dimension $n-d$. Hence and according to Theorem 2, one can construct it in time $\mathcal{O}((n-d)^2 d^3)$. However, we will show how to make use of the special form of K_n to construct it in time $\mathcal{O}(d(n-d))$.

In fact, we can construct this orthogonal lattice incrementally, i.e., from K_n, we will easily derive K_{n+1}. Indeed, let $\mathcal{K} = (k_1, \ldots, k_{n-d})$ be a basis of K_n. It is clear that $(k_i, 0) \in K_{n+1}$. 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 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 $\sum_{0 \leq i \leq d-1} -m_{i,n} x^i + \sum_{d \leq i \leq n-1} 0 \cdot x^i + x^n \equiv 0 \bmod f$, 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 K_{n+1} and linearly independent of the vectors $(k_i, 0)$. Since $\dim(K_{n+1}) = \dim(K_n) + 1$, we suggest the following: if $\mathcal{K}_n = (k_1, \ldots, k_{n-d})$ is a basis of K_n then $\mathcal{K}_{n+1} = (k'_1, \ldots, k'_{n+1-d})$ is a basis of K_{n+1} where $k'_i = (k_i, 0)$ for $1 \leq i \leq n-d$ and $k'_{n+1-d} = u$. We derive then the following algorithm to compute K_n:

Input : The lattice $\mathcal{L}(M_n)$ given by the matrix $M_n = (m_{i,j})$, where $0 \leq i < d$ and $0 \leq j < n$.
Output : The orthogonal lattice $k_n = M_n^{\perp}$.
Create the matrix $\mathcal{B}(K_n) = (k_{i,j})$, $0 \leq i < n - d$ 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 the matrix $\mathcal{B}(K_n)$ form a basis of the orthogonal lattice K_n;

Algorithmus 2 : **Computing the orthogonal lattice** K_n

The output matrix representing the orthogonal lattice K_n will have the following shape:

$$\mathcal{B}(K_n) = \begin{pmatrix} -m_{0,d} & \dots & -m_{d-1,d} & 1 & 0 & 0 & \dots & 0 \\ -m_{0,d+1} & \dots & -m_{d-1,d+1} & 0 & 1 & 0 & \dots & 0 \\ \vdots & & \vdots & \vdots & & \ddots & & \\ -m_{0,n-1} & \dots & -m_{d-1,n-1} & 0 & 0 & \dots & 0 & 1 \end{pmatrix}$$

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

Proof. It is clear that Algorithm 2 runs in time $\mathcal{O}(d(n-d))$. It remains to prove that it actually computes a basis of the orthogonal lattice K_n.

Since the lattice K_n has dimension $n - d$ according to Theorem 1 and the output matrix has $(n - d)$ rows, it suffices to prove that these rows form a generating family of K_n. Let $v = (v_0, \dots, v_{n-1}) \in K_n$. Then by definition of the orthogonal lattice, $< b, u >= 0$ for all vectors u in the lattice $\mathcal{L}(M_n)$, for instance $< b, u_i >= 0$ for all the lattice basis vectors u_i, rows of the matrix M_n. It follows that $\sum_{0 \leq j \leq n-1} m_{i,j} b_j = 0$ and thus $b_i = -\sum_{d \leq j \leq n-1} m_{i,j} b_j$ for $0 \leq i \leq d-1$. Consequently,

$$\begin{aligned} b &= (b_0, \dots, b_n) \\ &= (-\sum_{d \leq j \leq n-1} m_{0,j} b_j, \dots, -\sum_{d \leq j \leq n-1} m_{d-1,j} b_j, b_d, \dots, b_{n-1}) \\ &= \sum_{d \leq j \leq n-1} b_j(-m_{0,j}, \dots, -m_{d-1,j}, 0, \dots, 1, 0, \dots, 0). \end{aligned}$$

b can then be written as a linear combination of the rows of the resulting matrix $\mathcal{B}(K_n)$ with coefficients b_j, $d \leq j \leq n-1$, which concludes the proof.

□

Piecing all together. We are interested in the lattice L_n, set of all multiples over $\mathbb{Z}_2$, of the polynomial f, that have degree at most $n - 1$. As mentioned earlier, a generating family of this lattice is given by a basis of the lattice K_n and the already mentioned vectors v_i, $1 \leq i \leq n$:

$$v_i(j) = \begin{cases} 2 \text{ if } j = i, \\ 0 \text{ otherwise.} \end{cases}$$

where $v_i(j)$ denotes the j-th entry of the n-length vector v_i. Let $k_1, \ldots, k_{n-d}$ be the rows of the matrix $\mathcal{B}(K_n)$ computed by Algorithm 2. The vectors v_i, $d+1 \leq i \leq n$ can be generated by the vectors v_i, $1 \leq i \leq d$ and the vectors $k_1, \ldots, k_{n-d}$. In fact, for all $d+1 \leq i \leq n$:

$$v_i = 2 \cdot k_i - \sum_{1 \leq j \leq d} k_i(j) v_j.$$

Therefore, a generating family of L_n consists simply of the vectors $v_1, \ldots, v_d, k_1, \ldots, k_{n-d}$. The algorithm to compute a basis for L_n follows in a straightforward way:

Input : The polynomial f, over $\mathbb{Z}_2$, of degree d, and a bound $n > d$

Output : A basis for the lattice L_n.

Build the matrix M_n whose columns are the $h_i = x^i \bmod f$ for $0 \leq i < n$;

Call the algorithm 2 on the input M_n to compute a basis for the orthogonal lattice K_n ;

K_n basis vectors together with the vectors $v_1 \ldots v_d$ form a basis for L_n;

Algorithmus 3 : **Computing the lattice** L_n

The matrix representing the lattice L_n, given that $M_n = (m_{i,j})$, $0 \leq i < d$, $0 \leq j < n$, is:

$$\mathcal{B}(L_n) = \begin{pmatrix} 2 & 0 & 0 & 0\,0 & 0 & \ldots 0 \\ 0 & 2 & 0 & 0\,0 & 0 & \ldots 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \\ 0 & \ldots & 2 & 0 & \ddots & \\ -m_{0,d} & \ldots & -m_{d-1,d} & 1\,0 & 0 & \ldots 0 \\ -m_{0,d+1} & \ldots & -m_{d-1,d+1} & 0\,1 & 0 & \ldots 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \\ -m_{0,n-1} & \ldots & -m_{d-1,n-1} & 0\,0 \ldots & 0 & 1 \end{pmatrix}$$

Lemma 3. *Algorithm 3 computes a basis of the lattice L_n with running time $\mathcal{O}(d(n-d))$.*

Proof. The lattice L_n has dimension n. From the discussion above, Algorithm 3 outputs a generating family for L_n whose number of elements is exactly n. Therefore, the algorithm computes a basis for L_n.

Concerning, the running time, the computation of K_n is done within $\mathcal{O}(d(n-d))$. It remains to figure out the cost of calculating the matrix M_n.

To compute $h_i = x^i \bmod f$, $0 \leq i < n$, we proceed as follows. First, we remark that for $0 \leq i < d$, we have $h_i = x^i \bmod f = x^i$. Next 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 $xh_{i-1} - \mathrm{LC}(h_i)f$ if ($\deg(h_{i-1}) = d-1$), where $\mathrm{LC}(p)$ refers to the leading coefficient of the polynomial p. Note that, thanks to the presence of the vectors $v_1, \ldots, v_d$, we can perform these computations over $\mathbb{Z}_2$ instead of $\mathbb{Z}$. This leads to an overall cost of $\mathcal{O}(d(n-d))$, which concludes the proof.

□

3.2 Finding the low weight polynomial multiples.

This is the most expensive part of Algorithm 1 since it corresponds to the basis reduction of the lattice L_n. The LLL reduction can be performed in $\mathcal{O}(n^6)$. In fact, 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 input basis vectors. In our case these have values in $\{0, \pm 1, 2\}$, thus $B = 1$. We conclude that the higher n, the more infeasible the attack gets.

Theorem 3. *Algorithm 1, in case the reduction applied is LLL, runs in $\mathcal{O}(n^6)$ arithmetic operations, and computes n multiples of the polynomial f of weight w_i: $w_i \leq 2^{n-1}\lambda_i(L_n)^2, 1 \leq i \leq n$, where $\lambda_i(L_n)$ denote the successive minima of the lattice L_n.*

Proof. We first show that Algorithm 1 computes multiples of f over $\mathbb{Z}_2$. Let $v = (v_0, \ldots, v_{n-1}) \in L_n$. Then $\sum_{j=0}^{n} v_j m_{i,j} = 0 \bmod 2$, $0 \leq i \leq d-1$. 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 [11] 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 2. In case the minimal weight $w = \min\{w_i \colon M_f(n, w_i) \neq \emptyset\}$ is greater than 4, then the successive minima of the lattice are simply the vectors $v_1, \ldots, v_n$. Therefore, Algorithm 1 computes multiples with weights $w_i \leq 2^{n+1}$. Note, that a reduced basis will always consist of at most d vectors whose corresponding polynomial is identically zero modulo 2 (the vectors v_i, $1 \leq i \leq d$) since otherwise it won't possible to generate the basis vectors of the orthogonal lattice K^n (the vectors k_i, $1 \leq i \leq n-d$). For instance a basis output by Algorithm 1 will never consist only of the vectors $v_1, \ldots, v_n$ since they don't span the lattice L_n. Therefore, Algorithm 1 will output at least $n - d$ "interesting" multiples, i.e., multiples that are not identically zero modulo 2.

Remark 3. In order to improve on the quality of the obtained basis, we could use, instead of the LLL reduction, Schnorr's reduction algorithm [21] or the recently improved algorithm [1]. We will obtain then approximation factors that are slightly sub-exponential , namely $2^{\mathcal{O}(n(\log\log n)^2/\log n)}$ and $2^{\mathcal{O}(n \log\log n/\log n)}$ respectively. Note that an exact solution of the lowest weight multiple (or even an approximation to within polynomial factors in the dimension n), can be achieved in exponential running time. There exists also a heuristic that estimates the vectors lengths in a reduced basis output by Algorithm 1 by the product of the square root of the dimension n and the n-th root of the lattice determinant, that is 2^d. This gives us multiples of weights with approximation factors polynomial in n to the actual minimal weight.
On the practical side, the LLL algorithm, despite its pessimistic theoretical bounds, achieves a basis with moderately short vectors.

Finally, to relate the quality of the basis and the sparseness of the resulting polynomials, it is easy to see that, due to the presence of the vectors $v_1, \ldots, v_n$, the vectors in a reduced basis, others than the vectors v_i's, will have coefficients in $\{0, \pm 1\}$. Thus, a short vector (in the sense of the ℓ_2-norm) produces a sparse polynomial since the weight of the resulting polynomial is simply the square of the vector's ℓ_2-norm.

3.3 Experiments

To validate our method, we tested it on some known polynomials, using the NTL library [23] developed by Victor Shoup on a 2.66-GHz Intel 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 [22] 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 [22]. Finally, it is worth noting that the best fully proved floating point arithmetic LLL variant is

due to Nguyen and Stehlé [17]. 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. $f = 1 + x^2 + x^4 + x^5 + x^6 + x^8 + x^9 + x^{10} + x^{11} + x^{13} + x^{14} + x^{15} + x^{17}$.

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	1	1	1	1	1	2	1
t	0	0	0	0	0	0	0	0	0	0.044	0.2

2. The Lattice method using the floating variant of LLL (BKZ_FP) from the NTL library with 300 for the block size and 100 for pruning (in dimension 513):

$n-1$	17	20	20	20	21	22	24	30	44	94	513
w_f	13	8	8	8	8	8	8	5	5	4	4
M	1	1	1	1	1	1	1	1	1	3	7
t	0	0	0	0	0	0	0	0	0	3.12	16.29

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	4	3
M	1	1	1	1	2	2	4	1	1	27	1
t	0	0.3	0.3	0.3	0.42	0.6	1.45	0.02	0.08	1.24	0.03

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 graph:

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 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 does not exist.
Next, we note that the coefficient α of LLL_FP and BKZ_FP is constant and about 1 and 2 respectively. This explains why the TMTO method looses the lead as soon as the weight w gets greater than 8.
We must however note that our lattice-based method, oppositely to the TMTO method, does not always give the sparsest multiples, especially when the dimension of the lattice is big (in the experiment above, we were not able to recover the 3-weight multiple in dimension 513). In fact, lattice basis reduction algorithms are known to give only approximations to the exact solutions, and the bigger the dimension of the lattice, the looser the approximation factors get. We discuss in the next section the limitations as well as the possible extensions of our method.

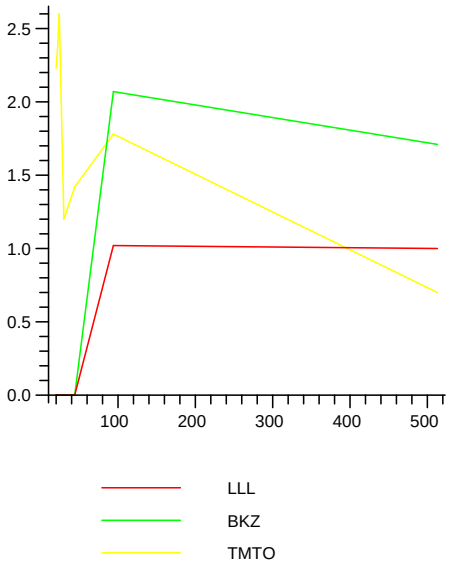

Fig. 1. Polynomial f

4 General Thoughts and Prospectives

4.1 Limitations

The strongest restriction on our algorithm is the lattice dimension, which corresponds to the bound on the polynomial multiple's degree. This is illustrated by the experiment discussed earlier. In fact, we were not able to find the 3-weight multiple of f, in dimension 513. This "non exactitude" of the solution becomes more tangible when the dimension of the lattice exceeds few dozens of hundreds. In fact, when we run our experiments on the E0 polynomial in dimension 2387, we got only a multiple with weight 41, given that at this dimension, there exists already a multiple with weight 31. This is again explained by the looseness of the solution which grows with the lattice dimension.

Furthermore, the reduced basis of L_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 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 in reducing the $(n+1)$-dimensional lattice $L'_n \subseteq \mathbb{Z}^{n+1}$ given by the basis $\mathcal{K}' = (l'_1, \ldots, l'_{n+1})$, where $l'_i = (l_i, 0)$, $1 \leq i \leq n$, and $l_{n+1} = (1, 0, \ldots, 0, 1)$. Experiments carried out improved slightly the results, for instance, we got a further multiple (with non zero constant term) with weight 4 at degree 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 its distance to the constant polynomial 1 . 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.

4.2 Possible extensions

An interesting question is to study the special form of the lattice L_n in order to reduce the cost of the reduction, or improve on the gotten results. In fact, the corresponding matrix in question is sparse, lower

triangular and has small entries. Hence, 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. We believe if one can "open up" the black box reduction algorithms and change them according to our special instance, one could possibly get better results.

Besides, with our approach, we managed to relate solving the low weight polynomial multiple problem to finding the shortest vector problem in L_n. We can also relate the closest vector problem to our problem, in fact, a lattice point in L_n 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 L_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 L_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.

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 , or even a ring $\mathbb{Z}_N$. However, the naive approach would not work since the correspondence short vector / sparse multiple won't hold anymore. The natural thing to do in this case is search for short l_2-norm polynomial multiples. Such a problem arises already in cryptanalysis, more precisely in Coppersmith's method for attacking RSA-based cryptosystems. Surprisingly, the algorithm used to solve this problem is a lattice-based technique.

5 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 $\mathcal{O}(n^6)$ in case LLL is the reduction algorithm used, and an experimental one about $\mathcal{O}(n)$. It takes then the lead as soon as the expected minimal weight w gets bigger than 8, provided that n is small. 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 confirmed our analysis by implementing the method using NTL; our method is applicable for relatively high dimensions (up to few hundreds), using the floating variants of LLL, and has proved very efficient in giving approximate solutions to instances that are intractable for 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. 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.), LNCS, vol. 1807, Springer, 2000, pp. 573,588.
4. A. Chose, P. Joux and M. Mitton, *Fast correlation attacks: an algorithmic point of view*, Advances in Cryptology - EUROCRYPT 2002 (L. R. Knudsen, ed.), LNCS, vol. 2332, Springer, 2002, pp. 209,221.
5. 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, p. to appear.
6. P. Ekdahl and T Johansson, *Distinguishing Attacks on SOBER-t16 and t32*, Advances in Cryptology - FSE 2002 (Joan Daemen and Vincent Rijmen, eds.), LNCS, vol. 2365, Springer, 2002, pp. 210,224.
7. R.G. Gallager, *Low-density parity-check codes.*, IRE Trans. Inform. Theory **IT-8** (1962), 21–28.
8. T. Johansson and F. Jönsson, *Fast correlation attacks based on turbo code techniques codes*, Advances in Cryptology - CRYPTO 1999 (M. J. Wiener, ed.), LNCS, vol. 1666, Springer, 1999, pp. 181,197.
9. ———, *Improved fast correlation attack on stream ciphers via convolutional codes*, Advances in Cryptology - EUROCRYPT 1999 (J. Stern, ed.), LNCS, vol. 1592, Springer, 1999, pp. 347,362.
10. ———, *Fast correlation attacks through reconstruction of linear polynomials*, Advances in Cryptology - CRYPTO 2000 (M. Bellare, ed.), LNCS, vol. 1880, Springer, 2000, pp. 300,315.
11. H. Lenstra, A. Lenstra and L. Lovasz, *Factoring polynomials with rational coefficients*, Math. ann **261** (1982), no. 4, 515–534.
12. Y. Lu and S. Vaudenay, *Faster correlation attack on Bluetooth keystream generator E0*, Advances in Cryptology - CRYPTO 2004 (M. K. Franklin, ed.), LNCS, vol. 3152, Springer, 2004, pp. 407,425.
13. W. Meier and O. Staffelbach, *Fast correlation attacks on certain stream ciphers*, J. Cryptology (1989), 159–176.
14. 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.
15. D. Micciancio and S. Goldwasser, *Complexity of lattice problems - A cryptographic perspective.*, Kluwer Academic Publishers, 2002.
16. P Nguyen, *La géométrie des nombres en cryptologie*, Ph.D. thesis, Laboratoire d'Informatique de l'École Normale Supérieure, France, November 1999.
17. P. Nguyen and D. Stehlé, *Floating-Point LLL Revisited*, Advances in Cryptology - EUROCRYPT 2005 (R. Cramer, ed.), LNCS, vol. 3494, Springer, 2005, pp. 215–233.
18. 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.), LNCS, vol. 1294, Springer, 1997, pp. 198,212.
19. ———, *The Béguin-Quisquater server-aided RSA protocol from Crypto '95 is not secure* , Advances in Cryptology - ASIACRYPT 1998 (K. Ohta and D. Pei, eds.), LNCS, vol. 1514, Springer, 1998, pp. 372–379.
20. ———, *Cryptanalysis of a fast public key cryptosystem presented at SAC '97*, SAC 1998 (Stafford E. Tavares and Henk Meijer, eds.), LNCS, vol. 1556, Springer, 1999, pp. 213–218.
21. C.-P. Schnorr, *A hierarchy of polynomial time lattice basis reduction algorithms*, Theoretical Computer Science **53** (1987), no. 2-3, 201–224.
22. 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.
23. V. Shoup, *NTL: a library for doing number theory.*, Available online at `http://www.shoup.net/ntl/`.
24. T. Siegenthaler, *Decrypting a class of stream ciphers using ciphertext only.*, IEEE Trans.Computers **C-34** (1985), no. 1, 81–84.
25. J. von zur Gathen and M. Nöcker, *Polynomial and normal bases for finite fields*, J.Cryptology **18** (2005), no. 4, 337–355.
26. D. Wagner, *A Generalized Birthday Problem.*, Advances in Cryptology - CRYPTO 2002 (M. Yung, ed.), LNCS, vol. 2442, Springer, 2002, pp. 288–304.