A New Approach for finding Low-Weight Polynomial Multiples

Laila El Aimani

Cadi Ayyad University
laila.elaimani@gmail.com

Abstract. We consider the problem of finding low-weight multiples of polynomials over binary fields; a problem which arises in stream cipher cryptanalysis or in finite field arithmetic. We first devise memory-efficient algorithms based on the recent advances in techniques for solving the knapsack problem. Then, we tune our algorithms using the celebrated Parallel Collision Search (PCS) method to decrease the time cost at the expense of a slight increase in space. Both our memory-efficient and time-memory trade-off algorithms improve substantially the state-of-the-art.

Keywords: Low-weight polynomial multiple · Stream cipher cryptanalysis · Knapsack · Collision-finding algorithm · Time-memory trade-off.

1 Introduction

We consider the following problem:

Definition 1 (The Low-Weight Polynomial Multiple (LWPM) problem). *Given a binary polynomial $P \in \mathbb{F}_2[X]$ of degree d and a bound n, find a multiple of P with degree less than n and with the least possible weight ω, where the weight of a multiple is the number of its nonzero coefficients.*

The LWPM arises in stream cipher cryptanalysis, and in efficient finite field arithmetic.

Stream cipher cryptanalysis. A Linear Feedback Shift Register (LFSR) is the core component of a large class of stream ciphers. It consists of an initial state, which corresponds to the shared secret key, and a connection polynomial over $\mathbb{F}_2$. A parity check for an LFSR is a multiple of its connection polynomial.

Fast correlation attacks [19,16] are well-known cryptanalytic attacks against LFSR-based stream ciphers. They recover the initial state of the constituent LFSR by viewing the output keystream as a noisy transmission of the sequence generated by the LFSR. The attacks take advantage of parity check equations satisfied by the LFSR output, which are given by multiples of the LFSR connection polynomial. To keep the bias as low as possible, low-weight multiples are required.

Another type of attacks against LFSR-based stream ciphers is *distinguishing attacks*, which aim at verifying whether a bitsream is encryption of some message. These attacks assume the keystream can be written as the sum of some γ-biased sequence and an LFSR's output. Given a parity check equation of the LFSR with weight ω, the output keystream is biased with $\frac{1}{2}\gamma^{\omega}$, which requires $\gamma^{-2\omega}$ samples to build the distinguisher according to standard cryptanalytic techniques. Thus the need for a low-weight multiple of the LFSR connection polynomial.

Finite field arithmetic. Another important application of low-weight multiples lies in representations of finite fields. In fact, von zur Gathen and Nöker[22] found that $\mathbb{F}_{2^d} = \mathbb{F}_2[x]/(g)$, where g is a low-weight irreducible polynomial of degree d, is the most efficient representation of finite fields if exponentiation is a core operation. Ideally, one would use irreducible polynomials of weight 3. However, these do not always exist. Brent and Zimmerman [3] proposed an interesting solution: take an irreducible polynomial $f \in \mathbb{F}_2[X]$ of degree d but possibly large weight, a multiple g of f with small weight, and work in the ring $\mathbb{F}_2[X]/(g)$ most of the time, going back to the field $\mathbb{F}_{2^d}$ only when necessary.

1.1 Related work

There have been several approaches for computing low-weight multiples of polynomials. Most methods first estimate the minimal possible weight ω of multiples of the given polynomial P with degree at most n, then look for multiples of weight at most ω. To estimate the minimal weight, one solves for ω_e the following inequality

$$\binom{n}{\omega_e} \geq 2^d \tag{1}$$

where d is the degree of P; the minimal weight ω is the smallest solution. In fact, if multiples are uniformly distributed, then one expects the inequality to hold. It is worth noting that the number of such multiples can be approximated by $\mathcal{N}_M = 2^{-d}\binom{n}{\omega}$.

Given a polynomial $P \in \mathbb{F}_2[X]$ of degree d and a bound n, we summarize below the strategies used to find a multiple of P of degree at most n and with the least possible weight ω. We describe the time or space complexity using the Big-O notation, which denotes the worst case complexity of the algorithms. Also, we use the approximation $\binom{n}{\omega} \approx O(n^{\omega})$.

Discrete-log-based techniques They were introduced in [17], then improved and generalized in [8,18]. They work with discrete logarithms in the multiplicative group of $\mathbb{F}_{2^d}$ instead of the direct representation of the polynomials. [8] use a time-memory trade-off to solve the problem in time $O(n^{\lceil \frac{\omega-2}{2} \rceil})$ and memory $O(n^{\lfloor \frac{\omega-2}{2} \rfloor})$. [18] provide an efficient-memory algorithm that runs in approximately $O(\frac{2^d}{n})$. The methods assume however a constant cost of the discrete logarithm computations, using precomputed tables that do not require excessive storage. This is not the case if $2^d - 1$ is not smooth. Also, the methods assume some conditions on the input polynomial: primitive in case of [8] or product of powers of irreducible polynomials with coprime orders in case of [18].

Syndrome decoding This technique reduces LWPM to finding a low-weight codeword in a linear code; a popular problem for which there exists known algorithms to solve it, e.g. the so-called information-set decoding algorithms [20,6,2,14,13,15]. These algorithms introduce many parameters to optimize the running time and the memory consumption according to the problem instance, however, we can approximate the running time by $O(\mathrm{Poly}(n) \cdot (\frac{n}{d})^{\omega})$, and the memory complexity by $O(d^{\omega})$.

Lattice-based techniques This technique, introduced in [9], reduces the LWPM problem to finding short vectors in an n-dimensional lattice. The method uses the LLL reduction [12] to solve the problem in time $O(n^6)$ and space $O(n \cdot d)$. Unfortunately, this technique gives inaccurate results, i.e. fails to find a multiple with the least possible weight, as soon as the bound n exceeds few hundreds.

Birthday techniques This is by far the standard method for solving the LWPM problem. There exists a plethora of variations and improvements to this method. The standard approach works as follows. Set $\omega = q_1 + q_2 + 1$ and build a list H_1 of all weight-q_1 combinations of residues $X^i \bmod P$, $0 < i \leq n$ and a list H_2 of all weight-q_2 combinations of the same residues. Then look for pairs in the lists that sum to 1. Clearly this method runs in $O(n^{q_2})$ (if H_1 is implemented as an efficient hash table), and uses $O(n^{q_1})$ of memory. The usual time-memory trade-off uses $q_1 = \lfloor \frac{\omega-1}{2} \rfloor$ and $q_2 = \lceil \frac{\omega-1}{2} \rceil$ in order to balance the time and the space complexities.

Chose et al. [7] cut down the memory utilization to $O(n^{\lfloor \frac{\omega-1}{4} \rfloor})$ using a *match-and-sort* approach. Using a divide-and-conquer technique, the task of finding collisions in a search space of n^{ω} is divided into smaller tasks: find less restrictive collisions on smaller subsets, sort the results and then aggregate these intermediate results to solve the complete task.

Canteaut and Trabbia [5] introduced a memory-efficient method for solving the LWPM problem. They compute all residues $X^i \bmod P$, $0 \leq i \leq n$ and store the exponent i in a table indexed by $X^i \bmod P$. Then, they form all weight-$(\omega - 1)$ combinations of the residues and look for collisions with $X^j \bmod P$. Clearly $X^j + X^{i_1} + \cdots + X^{i_{\omega-1}}$ is a weight-ω multiple of P. The algorithm runs in $O(n^{\omega-1})$ and requires only linear memory.

When the degree of the multiple gets very large and there are many low-weight multiples, but it is sufficient to find only one, *Wagner's generalized birthday paradox* becomes more efficient. For instance, if

$n \geq 2^{d/(1+\log_2(\omega-1))}$, then this method finds a weight-ω multiple of P of degree at most n in $O((\omega-1)n)$ and uses $O(n)$ memory.

We summarize in Tables 1 & 2 the costs (time and space) of the different methods.

Method	Exhaustive search	Discretelog method	Birthday method	Lattice method
Time	$\min(2^{n-d}, n^{\omega})$	$\frac{2^d}{n}$	$n^{\omega-1}$	n^6

Table 1. Summary of memory-efficient techniques

Method	Discrete log technique	Syndrome Decoding	Birthday Paradox	Generalized BP
Time	$n^{\lceil \frac{\omega-2}{2} \rceil}$	$\text{Poly}(n) \cdot \left(\frac{n}{d}\right)^{\omega}$	$n^{\lceil \frac{\omega-1}{2} \rceil}$	$(\omega-1)2^{d/(1+\log_2(\omega-1))}$
Space	$n^{\lfloor \frac{\omega-2}{2} \rfloor}$	d^{ω}	$n^{\lfloor \frac{\omega-1}{4} \rfloor}$	$2^{d/(1+\log_2(\omega-1))}$

Table 2. Summary of time-memory trade-off techniques

1.2 Our Approach

We view the LWPM problem as a special instance of the following subset sum problem:

Definition 2 (Group Subset Sum Problem). *Let $(G, .)$ be an abelian group. Given $a_0, a_1, \dots, a_n \in G$ together with $\omega, 0 < \omega \leq \frac{n}{2}$ such that there exists some solution $\mathbf{z} = (z_1, \dots, z_n) \in \{0,1\}^n$ satisfying*

$$\prod_{i=1}^{n} a_i^{z_i} = a_0 \quad \text{with} \quad \text{weight(z)} = \omega$$

The goal is to recover $\mathbf{z}$ (or some other weight-ω solution $\mathbf{z}$).

This definition generalizes that in [10] as it does not impose the group order to be of bitsize n. It captures then the LWPM problem as follows. Let P be a degree-d polynomial in $\mathbb{F}_2[X]$. Consider further the group $(\mathbb{F}_2^d, +)$ of d-dimensional vectors over $\mathbb{F}_2$, where the group law is the bitwise addition over $\mathbb{F}_2$. A weight-ω multiple $1 + \sum_{i=1}^{n} z_i X^i$ of P, with nonzero constant term and degree at most n satisfies:

$$\sum_{i=1}^{n} z_i a_i = a_0 \quad \text{with } \mathrm{a_i = X^i \bmod P,} \ \ 0 \leq \mathrm{i} \leq \mathrm{n}$$

Note that the condition on the weight ($\omega \leq \frac{n}{2}$) is not restrictive as most instances of LWPM that arise from either stream cipher cryptanalysis or finite field arithmetic satisfy it. Actually, the searched weight ω is obviously smaller than the weight of P, which is often smaller than $\frac{d}{2}$, and thus smaller than $\frac{n}{2}$.
Also, for convenience purposes, we consider throughout the document the relative weight $\omega_n = \omega/n$.

The (group) subset sum problem is one of the most popular and ubiquitous problems in cryptography. It has undergone an extensive analysis with a focus on polynomial-memory algorithms to solve it. In fact, it is known that random-access memory is usually more expensive than time. Most algorithms for solving the subset sum problem [1,10] try to find as many representations as possible of the solution; in fact, the more representations there exists the faster the solution can be found. For example, the folklore algorithm, described in [11], represents the solution $z = x \parallel y$ as a concatenation of two $\frac{n}{2}$-dimensional vectors x and y with weight(x) = weight(y) = $\frac{\omega}{2}$. In the same spirit, [1] split the solution z into two n-dimensional vectors x and y , with weight(x) = weight(y) = $\frac{\omega}{2}$, that add up to z. Recently, [10] further increase the number of representations by splitting z into a sum *over* $\mathbb{Z}$ of two integers of smaller weight by exploiting the carry propagation.

In this paper, we adapt these ideas to our setting and achieve algorithms for LWPM that significantly improve the state-of-the-art. More precisely, we make the following contributions.

Contributions First, we present two memory-efficient algorithms for LWPM that improve the state-of-the-art in polynomial-memory algorithms for LWPM. The idea behind the algorithms consists in splitting the solution z into two n-dimensional vectors x and y that add up to z over $\mathbb{F}_2$. The weight of both x and y is some function of ω to be determined according to the input.

More precisely, Algorithm 1 assumes and puts in place a Bernoulli distribution on the representation of z, then determines the optimal weight $\phi(\omega)$ to be used for x and y. As a result, we significantly improve the running time offered by the state-of-the-art methods (see Figure 1).

Since Algorithm 1 uses a pseudo-random number generator to establish the desired Bernoulli distribution, it incurs a slight overhead in the computations. Therefore, we reinforce our contribution with Algorithm 2 which does not use any assumptions; the result still substantively betters the state-of-the-art (see Figure 1).

We show the practicality of our technique with an implementation of the algorithms that confirm our theoretical estimates.

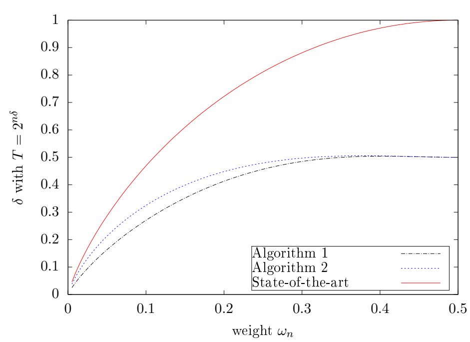
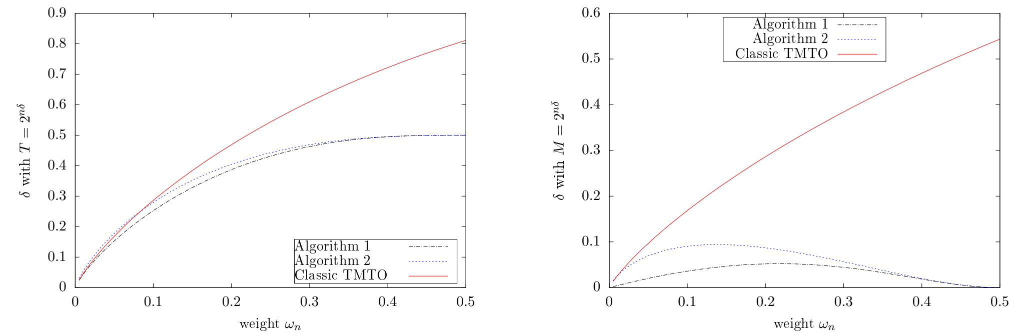


Fig. 1. Comparison between the efficient-memory techniques and our algorithms

Second, we tune our algorithms via the Parallel Collision Search (PCS) technique [21] to decrease the running time at the expense of memory. Again, we improve the classic Time-Memory Trade-off (TMTO) or birthday method, described earlier in the text, in both time and space (see Figure 2).

The rest of the paper is organized as follows. Section 2 recalls the necessary background and establishes the notation that will be used throughout the document. Sections 3 & 4 respectively describe, analyze, and experimentally validate our algorithms. Finally, the time-memory trade-off tuning of the proposed algorithms is given in Section 5.



(a) **Time complexity** (b) **Memory complexity**

Fig. 2. Comparison between the classic TMTO and our time-memory trade-off algorithms

2 Theoretical Background

2.1 Notations and Conventions

Let $a, b \in \mathbb{N}$ with $a < b$. We conveniently write $[a, b] := \{a, a+1 \ldots, b\}$. For a vector $z = (z_1, \ldots, z_n) \in \{0,1\}^n$, we denote by $\text{weight}(z) := |\{i \in [1, n] \colon z_i = 1\}|$. $\mathbb{Z}_N$ denotes the ring of integers modulo N. $\mathbb{F}_2$ denotes the field of two elements where the additive identity and the multiplicative identity are denoted 0 and 1, as usual. $\mathbb{F}_2[X]$ refers to the ring of polynomials with coefficients in $\mathbb{F}_2$.

Let $P \in \mathbb{F}_2[X]$. $\deg(P)$ and $\text{weight}(P)$ refer to the degree and weight of P respectively; the weight of a polynomial in $\mathbb{F}_2[X]$ corresponds to the number of its non-zero coefficients. In the text, we identify polynomials in $\mathbb{F}_2[X]$ with their coefficient vectors. For instance, the sum of two polynomials in $\mathbb{F}_2[X]$ is the sum over $\mathbb{F}_2$ of their coefficient vectors termwise.
Suppose $\deg(P) = d$. Then, $\mathbb{F}_2[X]/P$ denotes the ring of polynomials modulo P; addition and multiplication are performed modulo P. Finally, $(\mathbb{F}_2^d, +)$ refers to the group of d-dimensional vectors over $\mathbb{F}_2$, where the group law + is the bitwise addition and the identity is referred to as $0_{\mathbb{F}_2^d}$.

The Big-O, Θ, and $\tilde{\Theta}$ notations. The Big-O notation represents the upper bound of the running time of an algorithm; it gives then the worst case complexity of an algorithm. The Θ notation represents the upper and the lower bound of the running time of an algorithm. It is useful when studying the average case complexity of algorithms. The $\tilde{\Theta}$ notation suppresses the polynomial factors in the input. For example $\tilde{\Theta}(2^n)$ suppresses the polynomial factors in n.

Binomial coefficient. The binomial coefficient $\binom{n}{k}$ refers to the number of distinct choices of k elements within a set of n elements. We have:

$$\binom{n}{k} = \frac{n!}{k! \cdot (n-k)!}$$

Often, we need to obtain asymptotic approximation for binomials of the form $\binom{n}{\alpha\, n}$ or $\binom{n}{\lfloor \alpha\, n \rfloor}$ for values $\alpha \in]0, 1[$. This is easily achieved using Stirling's formula:

$$n! = (1 + o(1))\, \sqrt{2\pi n} \left(\frac{n}{e}\right)^n$$

Thus

$$\binom{n}{\alpha n} \approx \frac{1}{\sqrt{2\pi\ n\alpha(1-\alpha)}} \cdot 2^{nH(\alpha)}$$

where H is the binary entropy function defined as $H(x) := -x\log_2(x) - (1-x)\log_2(1-x)$; $\log_2$ is the logarithm in base 2. We can then write

$$\binom{n}{\alpha n} = \Theta\left(n^{-1/2}2^{nH(\alpha)}\right) \quad \text{or} \quad \binom{n}{\alpha n} = \tilde{\Theta}\left(2^{nH(\alpha)}\right)$$

Probability laws. For a finite set E, $e \in_R E$ refers to drawing uniformly at random an element e from E. The PMF of a random variable denotes its probability mass function.

Let X be a random variable, $p \in [0,1]$, and $n \in \mathbb{N}$.
$X \sim \text{Bernoulli}(p)$ signifies that X takes the value 1 with probability p and the value 0 with probability $1-p$.
$X := (X_1, \ldots, X_n) \sim \text{Bernoulli}(p,n)$ means that the X_i are independent and identically distributed with $X_i \sim \text{Bernoulli}(p)$, for $i \in [1,n]$.
$X \sim \text{Binomial}(p,n)$ means that X follows the Binomial distribution with PMF:

$$\Pr[X = k] = \binom{n}{k} p^k (1-p)^{n-k} \ , \ \ k \in [0,k]$$

Finally, if $X \sim \text{Bernoulli}(p,n)$, then the random variable Y corresponding to the number of successes of X follows the binomial distribution, i.e. $Y := \text{weight}(X) \sim \text{Binomial}(p,n)$.

2.2 Random Functions

Birthday paradox. Let E be a finite set of n elements. If elements are sampled uniformly at random from E, then the expected number of samples to be taken before some element is sampled twice is less than $\sqrt{\pi n/2} = \Theta(\sqrt{n})$. The element that is sampled twice is called a **collision** . See [11] for the details.

Expected number of collisions. Let $f : E \to F$ be a random function. We are interested in the expected number of collisions of f, i.e. the number of distinct pairs (x,y) with $f(x) = f(y)$. For instance, if k elements have the same value, this counts as $\binom{k}{2}$ collisions.

Fact 1 *Let $f : E \to F$ be a random function, with $|E| = n$ and $|F| = m$. The expected number of f collisions is $\Theta\left(\frac{n^2}{2m}\right)$.*

Proof. For each pair $\{x,y\}$ $(x \neq y)$, we define the following indicator variable:

$$I_{\{x,y\}} = \begin{cases} 1 & \text{if } f(x) = f(y) \\ 0 & \text{otherwise} \end{cases}$$

Let further C denotes the number of collisions of f. C is a random variable whose expectation $E(C)$ is given by

$$\begin{aligned} E(C) &= \sum_{\{x,y\}} E(I_{\{x,y\}}) \\ &= \frac{1}{m}\sum_{\{x,y\}} 1 = \frac{1}{m}\binom{n}{2} \\ &= \Theta\left(\frac{n^2}{2m}\right) \end{aligned}$$

□

Collision-finding algorithms Let $f : E \rightarrow F$, with $F \subseteq E$, be a random function. According to the birthday paradox, a collision of f can be found in roughly $\Theta(\sqrt{|F|})$ evaluations. Common search algorithms, e.g. Brent's cycle-finding algorithm [4], achieve this by computing a chain of invocations of f from a random starting point s until a collision occurs. In the text, the notation $(x, y) \longleftarrow \text{Rho}(f, s)$ refers to the collision (x, y) returned by f from starting point s, using a cycle-finding algorithm.

In [21], van Oorschot and Wiener extend this idea to search collisions between two functions f_1 and f_2 (both have the same domain E and range F). The construction defines a new function f that alternates between f_1 and f_2 depending on the input. The new function f is a random function, thus any cycle-finding algorithm applies and finds a collision for the new function in $\Theta(\sqrt{|F|})$ and constant memory. The found collision is a collision between f_1 and f_2 with probability $\frac{1}{2}$. Therefore the running time will roughly double if collisions are random. This is achieved by randomizing the output of the algorithm.

In fact, Brent's cycle-finding algorithm is likely to produce always the same collision. To remediate this problem, [1,10] consider a family of permutations $(P_k)_{k\in\mathbb{N}}$ in E addressed by k: they apply the collision-finding algorithm to $g\colon E \rightarrow E$ with $g(x) = P_k(f(x))$, where P_k is a random permutation from the considered family. In other terms, a new permutation is used with each invocation of the collision search algorithm, which ensures that the produced collisions are uniformly distributed.

3 First Algorithm

Let P be a d-degree polynomial over $\mathbb{F}_2$ with nonzero constant term, and $n > d$ be an integer. Our goal is to compute a multiple of P with the least possible weight, and with nonzero constant term and degree at most n. We proceed as follows.

We first determine the minimal weight using Inequality 1. Let ω be the found weight, and $1 + z = 1 + \sum_{i=1}^{n} z_i X^i$ be a weight-ω solution to the LWPM problem. We decompose z to $z = x + y$, with $x, y \in (\mathbb{F}_2^n, +)$ and $\text{weight}(x) = \text{weight}(y) = \phi = n * \phi_n$, where ϕ is a weight to be determined as a function of ω. Then, we compute x and y as a collision to a random function f, using any collision-finding algorithm, e.g. [4].
To compute ϕ, we assume a Bernoulli distribution on x and y. This assumption is plausible as the coordinates of x or y are independent (we ignore that they sum to ϕ, as this won't impact much the analysis). It will be then enough to have each coordinate (of x and y) equal 1 with the constant probability $\phi_n = \phi/n$.

This section is organized as follows. Subsection 3.1 defines the building blocks that will be used in the algorithm, namely the weight ϕ, the random function f and a further function that puts in place the Bernoulli distribution. Subsection 3.2 describes our first algorithm for solving LWPM. Finally Subsections 3.3 and 3.4 are dedicated respectively to the analysis and experimental validation of the presented algorithm.

3.1 Building blocks

Computation of ϕ. Assume a Bernoulli distribution on x and y. I.e. the coordinates of both x and y are considered independent trials with the constant probability of success $\Pr(x_i = 1) = \Pr(y_i = 1) = \phi_n = \frac{\phi}{n}$ for $i \in [1, n]$. We obviously ignore that $\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i = \phi$ as this won't impact much our analysis.
Therefore $z = x + y$ follows also a Bernoulli law with PMF $\Pr(z_i = 1) = 2\phi_n(1 - \phi_n)$, for $i \in [1, n]$. Moreover $\text{weight}(z) \sim \text{Binomial}(2\phi_n(1-\phi_n), n)$. Since $\text{weight}(z) = \omega - 1$, thus $\omega - 1 = 2n\phi_n(1-\phi_n)$, which is equivalent to $\phi_n = \frac{1}{2}(1 \pm \sqrt{1 - 2\omega_n})$, where $\omega_n := \frac{\omega-1}{n}$. Note that we assumed $\omega \leq \frac{n}{2}$, thus $\omega_n \leq \frac{1}{2}$.

Random function f. Let ϕ and ϕ_n be the quantities computed in the previous paragraph. Define the set $\mathcal{T}$:

$$\mathcal{T} = \left\{x \in \{0, 1\}^n \colon \text{weight}(x) = \phi = n * \phi_n\right\} \tag{2}$$

Let further $a_i = X^i \bmod P$ for $i \in [0, n]$. Consider the functions f_0, f_1:

$$\begin{gathered} f_0, f_1 \colon \mathcal{T} \longrightarrow \mathbb{F}_2^d \\ f_0(x) = \sum_{i=1}^{n} x_i a_i \ \text{ and } \ f_1(x) = a_0 + \sum_{i=1}^{n} x_i a_i \end{gathered} \tag{3}$$

Define further the function f:

$$f\colon \mathcal{T} \longrightarrow \mathbb{F}_2^d$$
$$x \longmapsto \begin{cases} f_0(x) & \text{if } h(x) = 0 \\ f_1(x) & \text{if } h(x) = 1 \end{cases} \tag{4}$$

where $h\colon \{0,1\}^n \to \{0,1\}$ is a random bit function. In other terms, f alternates between applications of f_0 and f_1 depending on the input. It is clear that a collision (x,y) of the function f will lead to a multiple of P with expected weight less than ω. In fact, a collision of type $f_i(x) = f_i(y),\ i = 0,1$ gives a multiple with expected weight $\omega - 1$, and a collision of type $f_i(x) = f_{1-i}(y),\ i = 0,1$ gives a multiple with expected weight ω.

Finally, since we will use a cycle-finding algorithm to search collisions of f, we need the function range and domain to be the same. To achieve this, we consider an injective map $\tau\colon \mathbb{F}_2^d \longrightarrow \mathcal{T}$ (provided $2^d \le |\mathcal{T}|$). Therefore, all collisions (x,y) of f satisfy

$$f(x) = f(y) \iff \tau \circ f(x) = \tau \circ f(y)$$

In this way, any cycle-finding technique can be applied to $\tau \circ f$ to search for collisions of f.
In the rest of the text, we conveniently identify $\tau \circ f$ with f; that is we assume that f outputs elements in $\mathcal{T}$, provided that $2^d \le |\mathcal{T}|$, but we keep in mind that $|f(\mathcal{T})| = 2^d$.

Bernoulli distribution on the input of f. Recall that function f inputs vectors of $\mathcal{T}$ that follow a Bernoulli distribution with parameters ϕ_n and n. That is, coordinates of the input vectors are independent and identically distributed with the constant probability ϕ_n of being equal to one. With this assumption, a collision of f leads to a multiple of P with expected weight less than ω.
We achieve such a distribution by using a random function σ

$$\sigma\colon \{0,1\}^n \longrightarrow \{0,1\}^n$$
$$x \longmapsto \sigma(x)\colon \sigma(x) \sim \text{Bernoulli}(\phi_n, n)$$

More precisely, σ uses the input elements as a seed to produce n-bit vectors that satisfy the Bernoulli distribution. Therefore, the input elements are only used to "remember" the state of the function, so that when it is called with the same value, it produces the same output.
σ has range $\{0,1\}^n$. In fact, for a random input element x in $\{0,1\}^n$, we have:

$$\Pr[\sigma(x) \in \{0,1\}^n] = \sum_{i=0}^{n} \binom{n}{i} \phi_n^i (1-\phi_n)^{n-i} = 1$$

Note however that σ outputs elements of weight ϕ with non-negligible probability:

$$\Pr[\sigma(x) \in \mathcal{T}, x \in_R \{0,1\}^n] = \binom{n}{\phi} \phi_n^{\phi} (1-\phi_n)^{n-\phi}$$
$$= \binom{n}{n\phi_n} 2^{-nH(\phi_n)} \approx \frac{1}{\sqrt{2\pi n \phi_n (1-\phi_n)}}$$

On other note, σ induces a uniform distribution on $\mathcal{T}$. In fact, let $y \in \mathcal{T}$ be a given element in $\mathcal{T}$, and x a random input element to σ

$$\Pr[\sigma(x) = y \mid \sigma(x) \in \mathcal{T}] = \frac{\Pr[\sigma(x) = y, \sigma(x) \in \mathcal{T}]}{\Pr[\sigma(x) \in \mathcal{T}]}$$
$$= \frac{\phi_n^{\phi}(1-\phi_n)^{n-\phi}}{\binom{n}{\phi}\, \phi_n^{\phi}(1-\phi_n)^{n-\phi}} = \frac{1}{|\mathcal{T}|}$$

Therefore, we conveniently assume in the rest of this section that σ has range $\mathcal{T}$ on which it induces a uniform probability distribution.

3.2 The algorithm

Consider the following map:

$$g\colon \{0,1\}^n \longrightarrow \mathcal{T}(\subset \{0,1\}^n)$$
$$x \longmapsto f \circ \sigma(x)$$

g is well defined as we assumed that σ has range $\mathcal{T}$. Moreover, g is a random function from $\{0,1\}^n$ to $\{0,1\}^n$, and thus we can apply any cycle-finding algorithm to search collisions for g. Note that σ will introduce some unnecessary collisions as we are only interested in collisions of f. We explain later how we compute this fraction of "useful" collisions among the total number of g collisions.

Now therefore, in consideration of the foregoing, a cycle-finding algorithm for g picks a random starting point $s \in_R \{0,1\}^n$, then computes a chain of invocations of g, i.e. $g(s), g^2(s) := g \circ g(s), \ldots$ until finding a repetition. If such a repetition leads to a valid collision (x, y), i.e. $g(x) = g(y)$ and $x \neq y$, return it otherwise start again with a new starting point. Termination of the algorithm is guaranteed if the execution paths from different starting points are independent. In other words, a random collision should be returned for each new starting point.

To randomize collisions, we introduce our last ingredient, a family of permutations P_k addressed by integer k:

$$P_k\colon \{0,1\}^n \longrightarrow \{0,1\}^n$$

The new function subject to collision search is

$$g^{[k]} = g \circ P_k\colon \mathcal{T} \longrightarrow \mathcal{T}$$

Note that the restriction of P_k to $\mathcal{T}$ is still a permutation from $\mathcal{T}$ to $P_k(\mathcal{T})(\subset \{0,1\}^n)$.

$g^{[k]}$ is a random function, with domain and range $\mathcal{T}$, which satisfies the randomness requirement on the computed collisions. In fact, for each new starting point s, a freshly random element $P_k(s)$ is obtained thanks to P_k (the permutation P_k is picked new with each new starting point), which is then used as a seed to σ to produce a random n-bit vector in $\mathcal{T}$ (with non-negligible probability) that satisfies the Bernoulli distribution. Therefore, execution paths, in cycle-searching algorithms for $g^{[k]}$, from different starting points are independent due to the application of a random permutation P_k with each new search.
Moreover, (x, y) is a collision for $g^{[k]}$ if and only if $(P_k(x), P_k(y))$ is a collision for g. Therefore, we can apply any cycle-finding algorithm to $g^{[k]}$ to search collisions for g.

We can now describe Algorithm 1 for solving the LWPM problem.

Remark 1. Algorithm 1 finds weight-ω multiples provided they exist. When Inequality 1 predicts a weight that does not exist, the algorithm runs indefinitely. As a safety valve, one can allow a margin in the breaking condition, and accept multiples with weights within that margin.

Remark 2. The μ_n's considered in the first loop are all less than $\frac{1}{2}$. In fact, they satisfy $\mu_n = 2\phi_n(1-\phi_n)$, and the function $x \longmapsto 2x(1-x)$ is upper bounded by $\frac{1}{2}$ for $x \in [0,1]$.

Remark 3. Both the values $\frac{1}{2}(1+\sqrt{1-2\mu_n})$ and $\frac{1}{2}(1-\sqrt{1-2\mu_n})$ for ϕ_n give the same expected time in terms of function calls, however, the latter value finds the solution faster as it is easier to manipulate sparse vectors.

3.3 Complexity analysis

Theorem 1. *Algorithm 1 runs in time* $\Theta(2^{C_t})$ *with*

$$C_t = \frac{d}{2} + n(-H(w_n) + H_1(\omega_n)) + \frac{3}{2}\log_2(2\pi n\omega_n(1-\omega_n))$$

where $H_1(\omega_n) = -\omega_n \log_2(2\omega_n(1-\omega_n)) - (1-\omega_n)\log_2(1-2\omega_n(1-\omega_n))$.

Algorithm 1 for LWPM

Input A polynomial P with degree d, and a bound n
Output A multiple M of P such that $\deg(M) \leq n$ and with the least possible weight.

```
Compute the expected minimal weight ω by solving Inequality 1
ω_n ⟵ (ω − 1)/n ; μ ⟵ ω − 1
repeat
    μ_n ⟵ μ/n; μ ⟵ μ + 1
    φ_n ⟵ 1/2 (1 ± sqrt(1 − 2 * μ_n)) ; φ ⟵ n * φ_n
until binom(n, φ) ≥ 2^d                      ▷ to ensure that f has range f(T) ⊆ T
repeat
    choose a random permutation P_k
    choose a random starting point s ∈_R T
    (x, y) ⟵ Rho(g^[k], s)
    (p, q) ⟵ (σ ∘ P_k(x), σ ∘ P_k(y))
    M ⟵ { X * (p + q)       if f_i(p) = f_i(q), i = 0, 1
        { 1 + X * (p + q)   if f_i(p) = f_{1−i}(q), i = 0, 1
until M ≡ 0 mod P and weight(M) ∈ [1, ω]
return M
```

We first note that $\omega - 1 = \phi$. In fact, ω is the smallest integer such that the inequality $\binom{n}{\omega-1} \geq 2^d$ holds. On other note, ϕ is the smallest integer such that $\binom{n}{\phi} \geq 2^d$, thus $\phi = \omega - 1$ and $\phi_n = \omega_n$.

Moreover, g and thus $g^{[k]}$ induces the uniform distribution on $g^{[k]}(\mathcal{T})$. In fact, σ induces the uniform distribution on $\mathcal{T}$, and f alternates with probability $\frac{1}{2}$ between applications of the deterministic functions f_0 and f_1. Thus, the birthday paradox applies and a collision of $g^{[k]}$ costs on average $2^{d/2}$. Actually, $g^{[k]}$ has domain $\mathcal{T}$ and range $g^{[k]}(\mathcal{T}) \subseteq T$, with $|g^{[k]}(\mathcal{T})| = 2^d$. Also, the expected number of $g^{[k]}$ collisions is $\Theta(\frac{|\mathcal{T}|^2}{2^{d+1}})$ according to Fact 1.

Proof. The algorithm searches collisions (x, y) for $g^{[k]}$ that correspond to f collisions, and that satisfy a weight condition. We call such collisions "useful collisions". Let $(x, y) \in_R \mathcal{T}^2$ with $(p, q) = (\sigma \circ P_k(x), \sigma \circ P_k(y))$. (x, y) is useful collision for $g^{[k]}$ if the following hold:

Event E_1**:** "$p, q \in \mathcal{T}$" (so that the function g and thus $g^{[k]}$ is well-defined)
Event E_2**:** "weight$(p + q) = n * \omega_n$"
Event E_3**:** "$X * (p + q)$ or $1 + X * (p + q)$ is a multiple of P"

Therefore the number of useful collisions is given by $|\mathcal{T}|^2 * \Pr[E1 \wedge E2 \wedge E_3]$.

According to the previous study of σ, we have $\Pr[E_1] \approx \frac{1}{2\pi n \phi_n (1-\phi_n)}$.

Moreover, $p \sim$ Bernoulli(ϕ_n, n) and $q \sim$ Bernoulli(ϕ_n, n). Therefore $p + q \sim$ Bernoulli$(2\phi_n(1 - \phi_n), n)$, and weight$(p + q) \sim$ Binomial$(2\phi_n(1 - \phi_n), n)$. Thus:

$$\begin{aligned}\Pr[E_2 \mid E_1] \approx \Pr[E_2] &= \binom{n}{n * \omega_n} (2\phi_n(1-\phi_n))^{n*\omega_n} (1 - 2\phi_n(1-\phi_n))^{n-n*\omega_n} \\ &= \binom{n}{\omega - 1} (2\phi_n(1-\phi_n))^{n*\omega_n} (1 - 2\phi_n(1-\phi_n))^{n-n*\omega_n}\end{aligned}$$

Finally, the probability that a random weight-ω polynomial with nonzero constant term and degree at most n equals a weight-ω multiple of P with nonzero constant term and degree at most n is $\binom{n}{\omega-1}^{-1} \mathcal{N}_M$, where $\mathcal{N}_M$ is the number of such multiples which equals $\binom{n}{\omega-1} 2^{-d}$.
Similarly, the probability that a random weight-$(\omega - 1)$ polynomial with zero constant term and degree at most n equals a weigh-$(\omega - 1)$ multiple of P with zero constant term and degree at most n is $\binom{n}{\omega-1}^{-1} \mathcal{N}'_M$, where $\mathcal{N}'_M$ is the number of such multiples which equals $\binom{n}{\omega-1} 2^{-d}$. Thus $\Pr[E_3 \mid E_2, E_1] = 2^{-d+1}$.

Since $\phi_n = \omega_n$ ($\phi = \omega - 1$), we conclude that the number of useful collisions is given by

$$\begin{aligned}
N_{\text{useful-collisions}} &= |\mathcal{T}|^2 * \Pr[E1 \wedge E2 \wedge E_3] \\
&\approx |\mathcal{T}|^2 2^{-d+1} \binom{n}{\omega - 1} (2\phi_n(1-\phi_n))^{n*\omega_n} (1 - 2\phi_n(1-\phi_n))^{n-n*\omega_n} \frac{1}{2\pi n \phi_n (1-\phi_n)} \\
&= |\mathcal{T}|^3 2^{-d+1} (2\omega_n(1-\omega_n))^{n*\omega_n} (1 - 2\omega_n(1-\omega_n))^{n-n*\omega_n} \frac{1}{2\pi n \omega_n (1-\omega_n)}
\end{aligned}$$

And the probability of a useful collisions is:

$$\begin{aligned}
\Pr[\text{useful} - \text{coll}] &= \frac{N_{\text{useful-collisions}}}{N_{\text{g}_\text{k}\text{-collisions}}} \\
&\approx \Theta\left(2^{-2} |\mathcal{T}| (2\omega_n(1-\omega_n))^{n*\omega_n} (1 - 2\omega_n(1-\omega_n))^{n-n*\omega_n} \frac{1}{2\pi n \omega_n (1-\omega_n)}\right) \\
&= \Theta\left(2^{nH(\omega_n)} (2\omega_n(1-\omega_n))^{n*\omega_n} (1 - 2\omega_n(1-\omega_n))^{n-n*\omega_n} \frac{1}{(2\pi n \omega_n (1-\omega_n))^{3/2}}\right)
\end{aligned}$$

Finally, the running time (in terms of function calls) of the algorithm is the product of $\Pr[\text{useful} - \text{coll}]^{-1}$ and the cost of a $g^{[k]}$-collision, i.e. $2^{d/2}$. Thus, on average, the running time exponent is approximately:

$$C_t = \frac{d}{2} + n(-H(w_n) + H_1(\omega_n)) + \frac{3}{2} \log_2(2\pi n \omega_n (1 - \omega_n)$$

where $H_1(\omega_n) = -\omega_n \log_2(2\omega_n(1-\omega_n)) - (1-\omega_n)\log_2(1 - 2\omega_n(1-\omega_n))$.

□

3.4 Experimental results

We run Algorithm 1 on the following polynomial P for $n \in [20, 600]$. The results are depicted in Figure 3.

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

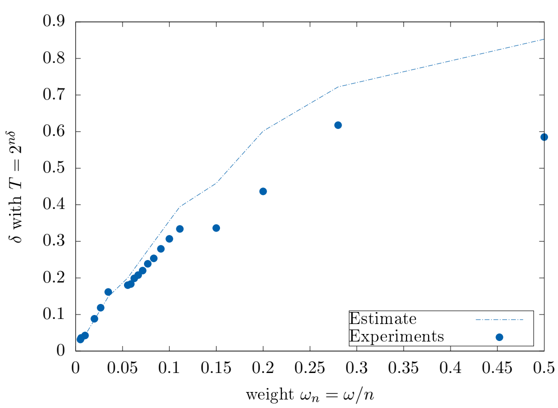


Fig. 3. Averaged function calls T for Algorithm 1 run on Polynomial P

We remark that the algorithm performs better for bigger weights ω. Actually, we did not consider in the analysis when function σ outputs vectors x with weights close to ϕ (smaller or bigger) that contribute to the solution; such an event is likely to occur when ω (and thus ϕ) is big.

4 Second Algorithm

Algorithm 1 in Section 3 incurs an overhead in the computations due to function σ. Actually, with each invocation of the function f, we make a call to σ which uses a pseudo-random number generator to establish the Bernoulli distribution on the input.

We remediate this problem in this section by not imposing anything on the input. Therefore, we decompose the solution z of LWPM into a pair (x, y), where x, y are n-bit vectors that do not enjoy any specific properties except having the same weight ϕ to be determined. We then look for such pairs by searching collisions of f.

Consider the set $\mathcal{T}$ defined in Statement 2, and let $x, y \in_R \mathcal{T}$. We proceed as follows. We first determine the PMF of the random variable $Y = \text{weight}(x + y)$ in order to compute ϕ as a function of ω. Then, we describe, analyze and experimentally validate our second algorithm in the subsequent subsections.

4.1 Computation of ϕ

Probability law of $Y = \text{weight}(x + y)$

Fact 2 $\Pr[Y = 2k + 1] = 0, \forall k \in \mathbb{N}$.

Proof. $\Pr[Y = 2k + 1]$ denotes the probability that x and y disagree on exactly $2k + 1$ positions. Let $\bar{x}$ and $\bar{y}$ be the $(2k + 1)$-bit strings extracted from x and y respectively, and composed of the bits where x and y disagree. Let further $x\backslash\bar{x}$ and $y\backslash\bar{y}$ be the remaining strings of x and y after extraction of $\bar{x}$ and $\bar{y}$ respectively. We have $\bar{x}_i = 1 - \bar{y}_i$, for $i \in [1, n]$. That is, there are $2k + 1$ ones distributed between the bits of $\bar{x}$ and $\bar{y}$.
Since $\text{weight}(x) = \text{weight}(y) = \phi$. Then, we will have $2\phi - 2k - 1$ ones distributed equally between the bits of $x\backslash\bar{x}$ and $y\backslash\bar{y}$ since $x\backslash\bar{x} = y\backslash\bar{y}$. This is impossible as $2\phi - 2k - 1$ is odd. We conclude that x and y cannot disagree on an odd number of positions. □

Fact 3 $\Pr[Y = k] = 0$, *for* $k \notin [0, \min(2\phi, n)]$.

Proof. There is a total of 2ϕ ones in both x and y. Therefore, x and y can disagree on at most 2ϕ positions. That is $\Pr[Y > 2\phi] = 0$. On other note, it is obvious that $\Pr[Y > n] = \Pr[Y < 0] = 0$. □

Let now, $k \leq \min(\phi, n/2)$ be an integer. $\Pr[Y = 2k]$ is given by the number of strings x and y that disagree on $2k$ positions, divided by the size of the probability space. The number of such strings is given by the product of:

– $\binom{n}{2k}$: the number of ways to choose the positions where x and y disagree.
– $\binom{2k}{k}$: the number of ways to distribute k ones in those $2k$ positions. In fact, let $\bar{x}$ and $\bar{y}$ be the $(2k)$-bit strings extracted from x and y respectively, and composed of the bits where x and y disagree. Then, $\bar{x}$ and $\bar{y}$ have the same weight, namely k, as x and y have the same weight ϕ, and agree on the remaining $n - 2k$ positions. Thus, the $2k$ ones must be equally distributed among $\bar{x}$ and $\bar{y}$.
– $\binom{n-2k}{\phi-k}$: the number of ways to choose $(n - 2k)$-bit strings with weight $(\phi - k)$. I.e. the number of strings where x and y agree.

The size of the probability space is given by $|\mathcal{T}|^2 = \binom{n}{\phi}^2$. Thus

$$\begin{aligned}\Pr[Y = 2k, k \leq \min(\phi, n/2)] &= \binom{n}{2k}\binom{2k}{k}\binom{n-2k}{\phi-k}\Big/\binom{n}{\phi}^2 \\ &= \binom{\phi}{k}\binom{n-\phi}{k}\Big/\binom{n}{\phi}\end{aligned}$$

We conclude that:

$$\Pr[\text{weight}(x + y) = 2k] = \begin{cases} \binom{\phi}{k}\binom{n-\phi}{k}\Big/\binom{n}{\phi} & \text{if } 0 \leq k \leq \min(\phi, n/2) \\ 0 & \text{otherwise} \end{cases}$$

Computation of ϕ Note that the PMF of $Y = \text{weight}(x + y)$ is reminiscent of the hypergeometric distribution G given by PMF:

$$\Pr[G = k] = \begin{cases} \binom{t}{k}\binom{n-t}{\phi-k}\Big/\binom{n}{\phi} & \text{if } 0 \le t, \phi \le n \text{ and } 0 \le k \le \min(\phi, t) \\ 0 & \text{otherwise} \end{cases}$$

and expectation $E(G) = \phi^2/n$. Actually, for $t = \phi$, we get

$$\Pr[G = k] = \begin{cases} \binom{\phi}{k}\binom{n-\phi}{\phi-k}\Big/\binom{n}{\phi} & \text{if } 0 \le \phi \le n \text{ and } 0 \le k \le \phi \\ 0 & \text{otherwise} \end{cases}$$

Therefore $\Pr[\text{weight}(x + y) = 2k] = \Pr[G = \phi - k]$. We derive the expectation of $Y = \text{weight}(x + y)$ as follows.

$$\begin{aligned} E(Y) &= \sum_{k=0,k=2p}^{2\phi} k \Pr[Y = k] = \sum_{k=0}^{\phi} 2k \Pr[Y = 2k] \\ &= \sum_{k=0}^{\phi} 2k \Pr[G = \phi - k] = 2 \sum_{k=0}^{\phi} (\phi - k) \Pr[G = k] \\ &= 2\phi - 2E(G) = 2\phi(1 - \phi/n) \end{aligned}$$

Therefore, if we conserve our previous notations: $\phi = n * \phi_n$, and $\omega - 1 = \omega_n * n$, and solve for ϕ_n the equation $\omega_n * n = 2\phi(1 - \phi/n)$. We get $\phi_n = \frac{1}{2}(1 \pm \sqrt{1 - 2\omega_n})$ $(\omega_n \le \frac{1}{2})$. Note that we get the same value we found for ϕ in Section 3, when we assumed a Bernoulli distribution on x and y, and consequently a binomial distribution on $\text{weight}(x+y)$ ($x+y \sim \text{Bernoulli}(\phi_n(1-\phi_n), n)$ and thus $\text{weight}(x+y) \sim \text{Binomial}(2\phi_n(1-\phi_n), n)$). This is not surprising; we know that for increasing n, the hypergeometric law converges to the binomial law.

4.2 The algorithm

Let (P, d, n) be a LWPM instance. We compute the minimal weight ω as usual by solving Inequality 1, then we compute ϕ_n as $\frac{1}{2}(1 \pm \sqrt{1 - 2(\omega - 1)/n})$ and ϕ as $n\phi_n$.
To compute a weight-ω multiple of P with degree less than n, we similarly search for collisions (p, q) of the function f defined earlier, where p and q are n-bit vectors with weight ϕ. There is a small particularity of this algorithm depending on the parity of ω. In fact, collisions of f are of two types:

Type 1 collisions that correspond to $f_i(p) = f_{1-i}(q)$, $i = 0, 1$. These collisions produce multiples of type $1 + X(p + q)$, with weight $1 + 2k$, $1 \le k \le \min(\phi, n/2)$.

Type 2 collisions that correspond to $f_i(x) = f_i(y)$, $i = 0, 1$. These collisions produce multiples of type $X(p+q)$, with weight $2k$, $1 \le k \le \min(\phi, n/2)$

Therefore, if $\omega = 1 + 2k$, we set $\mu =: \omega - 1$ and $\phi = n\phi_n$, with $\phi_n = \frac{1}{2}(1 \pm \sqrt{1 - 2\mu/n})$. As in Algorithm 1, we ensure that f outputs values in $\mathcal{T}$ (using the injective map $\tau\colon \mathbb{F}_2^d \longrightarrow \mathcal{T}$) by satisfying the condition $|\mathcal{T}| \ge 2^d$, where $|\mathcal{T}| = \binom{n}{\phi}$: we keep increasing μ until the inequality holds. Similarly, if $\omega = 2k$, then we initially set $\mu := \omega$ and keep increasing it until $\binom{n}{\phi} \ge 2^d$, where $\phi = n\phi_n$ and $\phi_n = \frac{1}{2}(1 \pm \sqrt{1 - 2\mu/n})$. We note again that both $\frac{1}{2}(1 + \sqrt{1 - 2\mu/n})$ and $\frac{1}{2}(1 - \sqrt{1 - 2\mu/n})$ lead to the same expected function calls, however, the latter value finds the solution faster as it is easier to manipulate sparse vectors.

Finally, to randomize collisions, it is enough to use any family of permutations $P_k\colon \mathcal{T} \longrightarrow \mathcal{T}$. The collision-finding algorithm is then applied to $f^{[k]} := P_k \circ f$.

We are now ready to give the pseudocode description of our second algorithm for LWPM in Algorithm 2.

First, we note that Remarks 1 & 2 & 3 for Algorithm 1 apply also here. Moreover, for even ω, Algorithm 2 finds multiples of the form $X * (p + q)$, where $p + q$ is a polynomial with degree at most $n - 1$. That is, the algorithm finds a weight-ω multiple with nonzero constant term and degree at most $n - 1$ (since P has

Algorithm 2 for LWPM

Input A polynomial P with degree d, and a bound n
Output A multiple M of P such that $\deg(M) \le n$ and with the least possible weight.

```
Compute the expected minimal weight ω by solving Inequality 1
if ω%2 = 1 then
    ω_n ⟵ (ω − 1)/n ; μ ⟵ ω − 1
else
    ω_n ⟵ ω/n ; μ ⟵ ω
end if
repeat
    μ_n ⟵ μ/n ; μ ⟵ μ + 1
    φ_n ⟵ 1/2 (1 ± sqrt(1 − 2 * μ_n)) ; φ ⟵ n * φ_n
until binom(n, φ) ≥ 2^d                    ▷ to ensure that f has range f(T) ⊆ T
repeat
    choose a random permutation P_k : T ⟶ T
    choose a random starting point s ∈_R T
    (p, q) ⟵ Rho(f^[k], s)
    M ⟵ { X * (p + q)      if f_i(p) = f_i(q), i = 0, 1
        { 1 + X * (p + q)  if f_i(p) = f_{1−i}(q), i = 0, 1
until M ≡ 0 mod P and weight(M) ∈ [1, ω]
return M
```

nonzero constant term) provided it exists. One could change, in this case, the definition of $\mathcal{T}$ and f and manipulate $(n+1)$-bit vectors instead of n-bit vectors in order to find multiples of degree at most n, but we opted for the above description to keep the algorithm simple.

4.3 Complexity analysis

Let $p, q \in_R \mathcal{T}$ and $j, \omega \in [1, n]$. Define the following events:

Event W: "weight$(p+q) = \omega$"
Event P_j: "$(p+q)_{1\ldots j} = \underbrace{0\ldots0}_{\text{j-1}}1$", where $(x)_{1\ldots j}$ denotes the length-j prefix of vector x.

Fact 4 *Let ω be an* even *weight in* $[1, n]$. *Then*

$$\Pr[W \wedge P_j] = \frac{\omega}{n-j+1}\Pr[W]\prod_{l=0}^{j-2}\left(1 - \frac{\omega}{n-l}\right)$$

Moreover, for small ω and i, with $j \le i \le n$:

$$\sum_{j=1}^{i}\Pr[W \wedge P_j] \ge i\Pr[W]\frac{\omega}{n-i+1}\left(\frac{n}{n-i+1}\right)^{\omega}$$

Proof. Let $\overline{P_j}$ denote the event "$(p+q)_{1\ldots j} = 0\ldots0$". We prove by induction that

$$\Pr[W \wedge \overline{P_j}] = \Pr[W]\prod_{l=0}^{j-1}\left(1 - \frac{\omega}{n-l}\right)$$

For $j = 1$:

$$\begin{aligned}\Pr[W \wedge (p+q)_1 = 0] &= \Pr[p_1 = q_1 = 0]\Pr[W \mid p_1 = q_1 = 0] + \Pr[p_1 = q_1 = 1]\Pr[W \mid p_1 = q_1 = 1] \\ &= \left(\frac{\binom{n-1}{\phi}}{\binom{n}{\phi}}\right)^2 \Pr[W \mid p_1 = q_1 = 0] + \left(\frac{\binom{n-1}{\phi-1}}{\binom{n}{\phi}}\right)^2 \Pr[W \mid p_1 = q_1 = 1] \\ &= (1-\phi_n)^2 \Pr[\text{weight}(p'+q') = \omega] + {\phi_n}^2 \Pr[\text{weight}(p''+q'') = \omega]\end{aligned}$$

Where, p', q' are random $(n-1)$-bit vectors with weight(p') = weight(p') = ϕ, and p'', q'' are random $(n-1)$-bit vectors with weight(p'') = weight(p'') = $\phi - 1$. Using the PMF of weight$(p+q)$, we compute $\Pr[\text{weight}(p'+q') = \omega]$ and $\Pr[\text{weight}(p''+q'') = \omega]$, and find that the expression of $\Pr[W \wedge (p+q)_1 = 0]$ simplifies to $\Pr[W]\left(1 - \frac{\omega}{n}\right)$.

Let now $j \geq 1$, and suppose the result holds true until j. We have

$$\Pr[W \wedge \overline{P_{j+1}}] = \Pr[W \wedge \overline{P_j} \wedge (p+q)_{j+1} = 0]$$

The event "$W \wedge \overline{P_j}$" is equivalent to the event W': weight$(p'+q') = \omega$", where p', q' are $(n-j)$-bit vectors such that weight(p') = weight(q') = ϕ_j with ϕ_j taking values in the interval $[\phi - j, \phi]$. Therefore:

$$\begin{aligned}\Pr[W \wedge \overline{P_{j+1}}] &= \Pr[W' \wedge (p'+q')_1 = 0] \\ &= \left(1 - \frac{\omega}{n-j}\right)\Pr[W'] = \left(1 - \frac{\omega}{n-j}\right)\Pr[W \wedge \overline{P_j}] \\ &= \Pr[W]\prod_{l=0}^{j}\left(1 - \frac{\omega}{n-l}\right)\end{aligned}$$

Since $\Pr[W \wedge P_j] = \Pr[W \wedge \overline{P_{j-1}}] - \Pr[W \wedge \overline{P_j}]$, then $\Pr[W \wedge P_j] = \frac{\omega}{n-j+1}\Pr[W]\prod_{l=0}^{j-2}\left(1 - \frac{\omega}{n-l}\right)$.

On the other hand, for small ω and i such that $j \leq i \leq n$, we have

$$\begin{aligned}\log_2\left(\sum_{j=1}^{i}\Pr[W \wedge P_j]\right) &= \log_2\left(\sum_{j=1}^{i}\frac{\omega}{n-j+1}\Pr[W]\prod_{l=0}^{j-2}\left(1 - \frac{\omega}{n-l}\right)\right) \\ &\geq \log_2\left(i\Pr[W]\frac{\omega}{n-i+1}\prod_{l=0}^{i-2}\left(1 - \frac{\omega}{n-l}\right)\right) \\ &= \log_2\left(i\Pr[W]\frac{\omega}{n-i+1}\right) + \sum_{l=0}^{i-2}\log_2\left(1 - \frac{\omega}{n-l}\right) \\ &\approx \log_2\left(i\Pr[W]\frac{\omega}{n-i+1}\right) + \sum_{l=0}^{i-2}\frac{\omega}{n-l} \\ &\approx \log_2\left(i\Pr[W]\frac{\omega}{n-i+1}\right) + \omega\left(\log_2(n) - \log_2(n-i+1)\right)\end{aligned}$$

The last equation is due to the approximation of the harmonic series $\sum_{k=1}^{n}\frac{1}{k} \approx \ln(n)$.

Finally: $\sum_{j=1}^{i}\Pr[W \wedge P_j] \geq i\Pr[W]\frac{\omega}{n-i+1}\left(\frac{n}{n-i+1}\right)^{\omega}$.

Theorem 2. *Algorithm 2 runs in time* $\tilde{\Theta}(2^{C_t})$ *where* $C_t = \frac{d}{2} + n\left(-H_2(\omega_n) + H(\omega_n)\right)$, *with* $H_2(\omega_n) = \omega_n + (1 - \omega_n)H\left(\frac{\omega_n}{2(1-\omega_n)}\right)$.

Proof. The algorithm searches for two types of f-collisions: **Type 1 collisions** when ω is odd, and **Type 2 collisions** when ω is even. We detail below the cost of each collision.

Type 1 collisions. A Type 1 collision (p, q) satisfies for an odd ω (i) weight$(p+q) = \omega - 1$ and (ii) $1 + X * (p+q)$ is a weight-ω multiple of P.
Define the following events for a pair $(p, q) \in_R \mathcal{T}^2$: W: "weight$(p+q) = \omega - 1$" and M: "$f \mid 1 + X * (p+q)$".
According to the probability law of weight$(p+q)$, we have

$$\Pr[W] = \binom{\phi}{(\omega-1)/2}\binom{n-\phi}{(\omega-1)/2}\Big/\binom{n}{\phi} = \binom{\omega-1}{(\omega-1)/2}\binom{n-\omega+1}{(\omega-1)/2}\Big/\binom{n}{\omega-1}$$
$$\approx 2^{n\left(\omega_n + (1-\omega_n)H\left(\frac{\omega_n}{2(1-\omega_n)}\right) - H(\omega_n)\right)} \frac{4(1-\omega_n)}{\sqrt{2\pi n \omega_n (2-3\omega_n)}}$$

In fact $\phi = \omega - 1$ (and thus $\phi_n = \omega_n$) since ϕ and $\omega - 1$ are the smallest integers that satisfy the inequality $\binom{n}{x} \geq 2^d$.

Further, and as argued previously, the probability that a random weight-ω polynomial with nonzero constant term and degree at most n equals a weight-ω multiple of P with nonzero constant term and degree at most n is $\binom{n}{\omega-1}^{-1}\mathcal{N}_M$, where $\mathcal{N}_M$ is the number of such multiples which equals $\binom{n}{\omega-1}2^{-d}$.
Therefore, for a pair $(p, q) \in_R \mathcal{T}^2$ and an odd ω

$$\Pr[(p,q) \text{ is a Type 1 collision}] = \Pr[W \wedge M] = \Pr[W]\Pr[M \mid W] = 2^{-d}\Pr[W]$$

This implies that we have heuristically $N_{\text{Type1-collisions}} = |\mathcal{T}|^2 2^{-d}\Pr[W]$ many Type 1 collisions. The probability $p_{\text{type1-collisions}}$ of finding such collisions is given by the ratio of $N_{\text{Type1-collisions}}$ and the total number of f collisions, estimated by $|\mathcal{T}|^2 2^{-d-1}$,

$$p_{\text{type1-collisions}} = \frac{|\mathcal{T}|^2 2^{-d}\Pr[W]}{|\mathcal{T}|^2 2^{-d-1}}$$
$$\approx \Theta\left(2^{n\left(\omega_n + (1-\omega_n)H\left(\frac{\omega_n}{2(1-\omega_n)}\right) - H(\omega_n)\right)} \frac{8(1-\omega_n)}{\sqrt{2\pi n \omega_n (2-3\omega_n)}}\right)$$

Each collision costs $\Theta(2^{d/2})$, therefore, the expected number of function calls before the algorithm terminates is $\Theta(2^{C_t}\text{Poly}_1(n))$:

$$C_t = \frac{d}{2} + n\left(-\omega_n - (1-\omega_n)H\left(\frac{\omega_n}{2(1-\omega_n)}\right) + H(\omega_n)\right) \text{ and } \text{Poly}_1(n) = \frac{\sqrt{2\pi n \omega_n (2-3\omega_n)}}{8(1-\omega_n)}$$

Type 2 collisions. When ω is even, the algorithm produces a Type 2 collision (p, q), characterized by: (i)weight$(p+q) = \omega$, (ii) $(p+q)_{1\ldots i} = 0\ldots01$, where i is the largest integer such that there exists a weight-ω multiple of P with nonzero constant term and degree $n - i$, and (iii) $X(p+q)$ is a weight-ω multiple of P of degree at most $n - i + 1$.
For a pair $(p, q) \in_R \mathcal{T}^2$, consider the events W and P_j defined earlier in this subsection, in addition to the event M: "$f \mid X * (p+q)$". Therefore

$$\Pr[(p,q) \text{ is a Type 2 collision}] = \sum_{j=1}^{i}\Pr[W \wedge P_j \wedge M] = \sum_{j=1}^{i}\Pr[W \wedge P_j]\Pr[M \mid W, P_j]$$

Again, the probability that a random weight-ω polynomial with nonzero constant term and degree $n - i$ equals a weight-ω multiple of P with nonzero constant term and degree $n - i$ is $\binom{n-i}{\omega-1}^{-1}\mathcal{N}'_M$, where $\mathcal{N}'_M$ is the number of such multiples which equals $\binom{n-i}{\omega-1}2^{-d}$. Therefore $\Pr[M \mid W, P_j] = 2^{-d}$ for $j \in [1, i]$. Furthermore, according to Fact 4, we have:

$$\Pr[(p,q) \text{ is a Type 2 collision}] = 2^{-d}\sum_{j=1}^{i}\Pr[W \wedge P_j] \quad \geq 2^{-d} i \Pr[W]\frac{\omega}{n-i+1}\left(\frac{n}{n-i+1}\right)^{\omega}$$

With

$$\Pr[W] = \binom{\phi}{\omega/2}\binom{n-\phi}{\omega/2}\Big/\binom{n}{\phi} = \binom{\omega-1}{\omega/2}\binom{n-\omega+1}{\omega/2}\Big/\binom{n}{\omega-1}$$

Using $\binom{n-1}{k} = \frac{n-k}{n}\binom{n}{k}$ and $\binom{n}{k-1} = \binom{n}{k}\frac{k}{n-k+1}$, we get:

$$\Pr[W] \approx \frac{(n-\omega+1)^2}{\omega(2n-3\omega+2)} \cdot 2^{n\left(\omega_n+(1-\omega_n)H\left(\frac{\omega_n}{2(1-\omega_n)}\right)-H(\omega_n)\right)} \cdot \frac{4(1-\omega_n)}{\sqrt{2\pi n\omega_n(2-3\omega_n)}}$$

By proceeding in the same way as for Type 1 collisions, we show that Algorithm 2 produces Type 2 collisions in $\Theta(2^{C_t}\mathrm{Poly}_2(n))$:

$$C_t = \frac{d}{2} + n\left(-H_2(\omega_n) + H(\omega_n)\right) \quad \text{with} \quad H_2(\omega_n) = \omega_n + (1-\omega_n)H\left(\frac{\omega_n}{2(1-\omega_n)}\right)$$

and

$$\mathrm{Poly}_2(n) = \frac{(2n-3\omega+2)}{(n-\omega+1)^2} \cdot \frac{\sqrt{2\pi n\omega_n(2-3\omega_n)}}{8(1-\omega_n)} \frac{n-i+1}{i} \left(\frac{n-i+1}{n}\right)^{\omega}$$

Note that $\left(\frac{n-i+1}{n}\right)^{\omega} \leq 1$, thus $\mathrm{Poly}_2(n)$ is indeed polynomial in n. □

4.4 Experimental results

We consider the same test polynomial in Subsection 3.4 for the same range of values $n \in [20, 600]$; the results are depicted in Figure 4. Note that we used the $\tilde{\Theta}$ notation for the estimated time, which explains the slight differences between the estimates and the experiments.

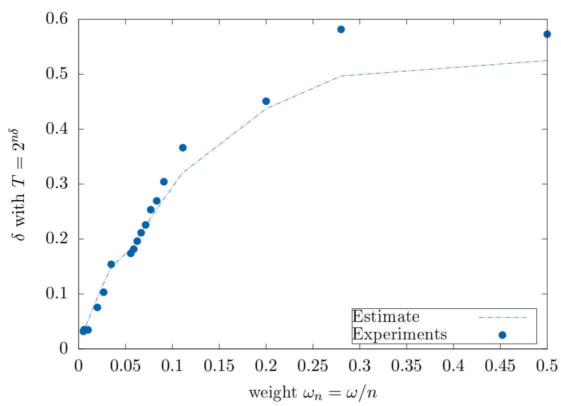


Fig. 4. Averaged function calls T for Algorithm 2

Before moving on to the next section, we compare the performance of our algorithms with existing memory-efficient methods for LWPM. According to Table 1, these lasts run in $\tilde{\Theta}(2^d)$ or $\tilde{\Theta}(2^{nH(\omega_n)})$. Actually, we discard the lattice method as it becomes inaccurate with increasing n (few hundreds). On other note, since ω is the smallest integer such that $\binom{n}{\omega-1} \geq 2^d$. We can assume that $2^d \approx \binom{n}{\omega-1} = \tilde{\Theta}(2^{nH(\omega_n)})$, where $\omega_n = \frac{\omega-1}{n}$. We conclude that existing memory-efficient methods for LWPM run in approximately $\tilde{\Theta}(2^{nH(\omega_n)})$. Using the same approximation, Algorithm 1 runs in $\tilde{\Theta}(2^{n(-\frac{H(\omega_n)}{2}+H_1(\omega_n))})$, whereas Algorithm 2 runs in $\tilde{\Theta}(2^{n(\frac{3H(\omega_n)}{2}-H_2(\omega_n))})$.

Figure 5 depicts the performance of our algorithms in comparison with the state-of-the-art methods. Note that our algorithms apply to any polynomial, and do not use any precomputed tables of discrete logarithms, unlike some existing memory-efficient methods (discrete-log-based ones).

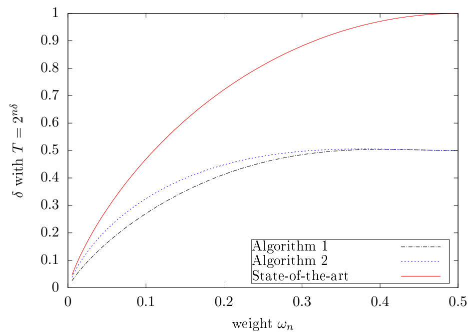


Fig. 5. Comparison between the efficient-memory techniques and our algorithms

5 Time-Memory Trade-off Variants

Our previously described algorithms allow fortunately for a time-memory trade-off, thanks to van Oorschot-Wiener's Parallel Collision Search (PCS) technique [21]. This technique has been extensively used in cryptanalysis since its introduction; it allows to efficiently find multiple collisions, of a random function, at a low amortized cost per collision. More precisely, let C be the time complexity to find a collision with polynomial memory, then PCS finds 2^m collisions in time $\tilde{\Theta}(2^{\frac{m}{2}}C)$ using $\tilde{\Theta}(2^m)$ memory.

In the following, we apply PCS to Algorithms 1 & 2 in order to decrease their time complexity at the expense of memory.

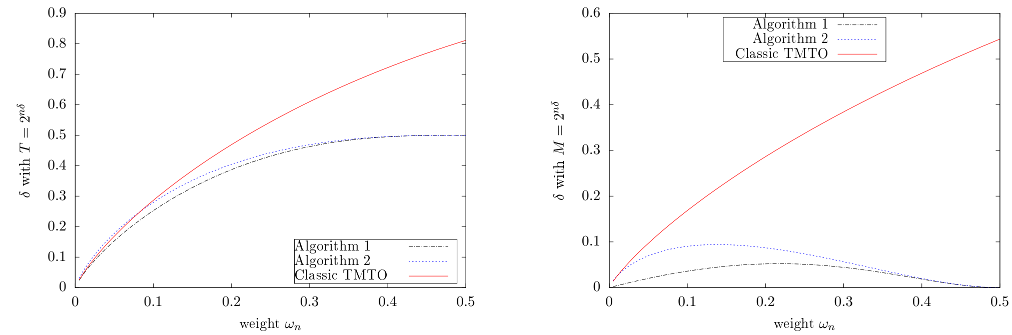
Algorithm 1 Trade-off. According to the analysis in section 3, Algorithm 1 requires to find $\tilde{\Theta}(2^{n(-H(w_n)+H_1(\omega_n))})$ collisions. In fact, this value corresponds to the number of examined collisions before coming across a so-called *useful collision*, i.e. a collision that leads to a solution to the LWPM problem. Each collision comes at the cost of $\tilde{\Theta}(2^{\frac{d}{2}})$. Therefore, using $M_{\text{tmto-1}} = \tilde{\Theta}(2^{n(-H(w_n)+H_1(\omega_n))})$ memory, the time complexity of the trade-off variant of Algorithm 1 reduces to $T_{\text{tmto-1}} = \tilde{\Theta}(2^{\frac{n(-H(\omega_n)+H_1(\omega_n))}{2}} \cdot 2^{\frac{d}{2}})$.
Again, since ω is the smallest integer such that $\binom{n}{\omega-1} \geq 2^d$. We can assume that $2^d \approx \binom{n}{\omega-1} = \tilde{\Theta}(2^{nH(\omega_n)})$, where $\omega_n = \frac{\omega-1}{n}$. Therefore $T_{\text{tmto-1}} \approx \tilde{\Theta}(2^{\frac{nH_1(\omega_n)}{2}})$.

Algorithm 2 Trade-off. Similarly, Algorithm 2 requires to find $\tilde{\Theta}(2^{n(H(\omega_n)-H_2(\omega_n))})$ collisions, each at the cost of $\tilde{\Theta}(2^{\frac{d}{2}})$. Therefore, using $M_{\text{tmto-2}} = \tilde{\Theta}(2^{n(H(\omega_n)-H_2(\omega_n))})$ memory, the time complexity of the trade-off variant of Algorithm 2 reduces to

$$\begin{aligned} T_{\text{tmto-2}} &= \tilde{\Theta}(2^{\frac{n(H(\omega_n)-H_2(\omega_n))}{2}} \cdot 2^{\frac{d}{2}}) \\ &\approx \tilde{\Theta}(2^{n(H(\omega_n)-\frac{H_2(\omega_n)}{2})}) \end{aligned}$$

We depict in Figure 6 the comparison between the trade-off variants of Algorithms 1 & 2, and the classical TMTO method whose time and memory complexity are $\tilde{\Theta}(2^{nH(\frac{\omega_n}{2})})$ and $\tilde{\Theta}(2^{nH(\frac{\omega_n}{4})})$ respectively.

Note that the time complexity depicted in Figure 6 upper estimates the time cost of our algorithms due to the upper approximation of d by $nH(\omega_n)$. In other words, our trade-off algorithms perform even better in practice. As an illustration, we provide in Figure 7 the comparison results for the polynomial P, used previously in the experiments.



(a) **Time complexity** (b) **Memory complexity**

Fig. 6. Comparison between the classic TMTO and our algorithms after applying PCS

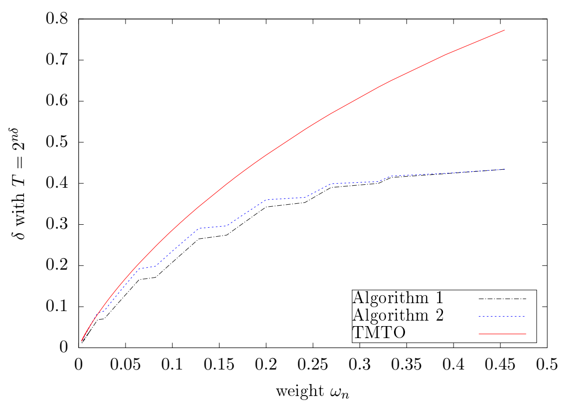


Fig. 7. Time complexity of the classic TMTO and our trade-off algorithms when applied to Polynomial P

References

1. Anja Becker, Jean-Sébastien Coron, and Antoine Joux, *Improved Generic Algorithms for Hard Knapsacks*, Advances in Cryptology - EUROCRYPT 2011 -, 2011, pp. 364–385.
2. Anja Becker, Antoine Joux, Alexander May, and Alexander Meurer, *Decoding Random Binary Linear Codes in $2^{n/20}$: How 1+1=0 Improves Information Set Decoding*, Advances in Cryptology - EUROCRYPT 2012 -, 2012, pp. 520–536.
3. R. P. 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), 2003.
4. Richard P. Brent, *An Improved Monte Carlo Factorization Algorithm*, BIT Numerical Mathematics (1980), 176184.
5. 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.
6. Anne Canteaut and Florent Chabaud, *A New Algorithm for Finding Minimum-Weight Words in a Linear Code: Application to McEliece's Cryptosystem and to Narrow-Sense BCH Codes of Length 511*, IEEE Trans. Inf. Theory (1998), 367–378.
7. 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.
8. 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.
9. L. El Aimani and J. von zur Gathen, *Finding Low Weight Polynomial Multiples Using Lattices* , Poster session of the LLL + 25 conference, 2007, Full version available at the Cryptology ePrint Archive, Report 2007/423.

10. Andre Esser and Alexander May, *Low Weight Discrete Logarithm and Subset Sum in $2^{0.65n}$ with Polynomial Memory*, Advances in Cryptology - EUROCRYPT 2020 -, 2020, pp. 94–122.
11. Steven D. Galbraith, *Mathematics of Public Key Cryptography*, Cambridge University Press, 2012.
12. H. Lenstra, A. Lenstra and L. Lovasz, *Factoring Polynomials With Rational Coefficients*, Mathematische Annalen **261** (1982), no. 4, 515–534.
13. Carl Löndahl and Thomas Johansson, *Improved Algorithms for Finding Low-Weight Polynomial Multiples in $F_2[x]$ And Some Cryptographic Applications*, Des. Codes Cryptogr. (2014), 625–640.
14. Alexander May, Alexander Meurer, and Enrico Thomae, *Decoding Random Linear Codes in $\tilde{\mathcal{O}}(2^{0.054n})$*, Advances in Cryptology - ASIACRYPT 2011 -.
15. Alexander May and Ilya Ozerov, *On Computing Nearest Neighbors with Applications to Decoding of Binary Linear Codes*, Advances in Cryptology - EUROCRYPT 2015, 2015, pp. 203–228.
16. W. Meier and O. Staffelbach, *Fast Correlation Attacks on Certain Stream Ciphers*, J. Cryptology (1989), 159–176.
17. W. T. Penzhorn and G. J. Kühn, *Computation of Low-Weight Parity Checks for Correlation Attacks on Stream Ciphers*, Cryptography and Coding, 1995, pp. 74–83.
18. Pietro Peterlongo, Massimiliano Sala, and Claudia Tinnirello, *A Discrete Logarithm-Based Approach to Compute Low-Weight Multiples of Binary Polynomials*, Finite Fields Their Appl. (2016), 57–71.
19. T. Siegenthaler, *Decrypting a Class of Stream Ciphers Using Ciphertext Only.*, IEEE Trans.Computers **C-34** (1985), no. 1, 81–84.
20. Jacques Stern, *A Method for Finding Codewords of Small Weight*, Coding Theory and Application, 1988, pp. 106–113.
21. Paul C. van Oorschot and Michael J. Wiener, *Parallel Collision Search with Cryptanalytic Applications*, J. Cryptol. (1999), 1–28.
22. J. von zur Gathen and M. Nöcker, *Polynomial and Normal Bases for Finite Fields*, J.Cryptology **18** (2005), no. 4, 337–355.