Multivariate Polynomial Inference in a Cryptographic Setting

Ramona Corbeanu[1], Diana Maimuţ[2], and George Teşeleanu[3]

[1] Department of Mathematics, University of Bucharest, Bucharest, Romania
ramona-georgiana.corbeanu@unibuc.ro
[2] Advanced Technologies Institute, Bucharest, Romania
diana.maimut@dcti.ro
[3] Simion Stoilow Institute of Mathematics of the Romanian Academy, Bucharest, Romania
george.teseleanu@yahoo.com

Abstract. In this paper, we generalize to the multivariate setting the current state-of-the-art methods in the literature for the inference of bivariate polynomials constructed recursively, by means of repeated additions and multiplications. We present two main approaches: the first one based on polynomial interpolation and the second one relying on lattice-based techniques for solving modular knapsack-type problems. Both the directions yield natural and practical generalizations, supported by detailed analyses of the underlying mathematical structures. Our methods can be useful for analysing the security of cryptographic algorithms, given their connection to basic operations serving as building blocks, for example in fully homomorphic encryption schemes.

Keywords: multivariate polynomial, Lagrange interpolation, modular knapsack problem, lattice reduction

1 Introduction

The problem of reconstructing polynomials by means of query-based black-box access was first investigated in [34] in the bivariate case. To the best of our knowledge, this direction did not benefit from a follow-up until the current study, remaining the only work directly addressing the recovery of polynomials constructed from repeated applications of two basic algebraic operations, namely addition and multiplication. Considering a cryptographic setting which can be linked to, *e.g.*, homomorphic encryption (HE), two approaches for bivariate polynomial inference were introduced in [34]: one based on Lagrange interpolation, and the other inspired by modular knapsack resolution techniques. In this paper, we focus on extending such methods to the multivariate setting, a challenge left explicitly as an open problem in [34]. We build upon the aforementioned approaches in order to tackle the central problem of our study, but deep-dive even further, providing the reader with more in-depth analyses and introducing a supplementary aggregation problem. In the interpolation-inspired case, our

attack scenario is based on repeated queries which enable the reconstruction of the polynomial of interest. For the modular knapsack paradigm, instead of taking advantage of many queries, we repeatedly call a preset modular knapsack resolution algorithm (in this work we focus on lattice-based methods).

To be more specific, our paper proposes different types of algorithms for recovering a multivariate polynomial P such that $P(a_1, \ldots, a_m) = r$ (for a given m), given the natural numbers $a_1, \ldots, a_m, r$. When passing to the multivariate setting, the main technical difficulty which we encountered was handling multiple modular reductions in order to reduce the problem to the two variables case. The reduction from m variables to 2 variables is achieved recursively. However, we had to overcome a specific issue: when reducing the problem from m to $m-1$ variables, it is necessary to ensure that the resulting polynomials remain in $\mathbb{Z}[x_1, \ldots, x_{m-1}]$.

In this context, we discuss related results that underpin our proposed methods within the relevant sections, namely multivariate polynomial interpolation in Section 3 and lattice-based modular knapsack algorithms in Section 4. Our proposed techniques demonstrate that it is possible to recover the exact polynomial implemented by a system, even when it is custom-designed. This observation highlights potential vulnerabilities and suggests that such algorithms should be more carefully protected, for instance by restricting the amount of accessible data or by increasing the density of the polynomial coefficients.

Motivation: Applications in Cryptography. In regards to cryptographic applications, our work can be seen as a continuation of [34], representing a further step toward the reverse engineering algorithms that rely on homomorphic operations over encrypted data. Hence, as a practical cryptographic application, we emphasize that our analysis is linked with the operations standing at the core of fully homomorphic encryption (FHE). The first fully homomorphic encryption (FHE) scheme was introduced by Gentry in [22], marking a major breakthrough. Another early FHE construction appeared in [16], followed by subsequent improvement (*e.g.* [6–11, 17, 21, 23, 32]). Note that our results are agnostic of FHE algorithms sub-categories like word-wise or bitwise. FHE became one of the key ingredients in privacy-enhancing cryptography, as mathematical guarantees can enhance the security environments, like healthcare and finance. For such critical applications, utility and confidentiality must coexist, as sensitive data needs be handled, enabling secure computation and analysis without the need to decrypt the underling information [15].

Another topic which is closely related to our work is Oblivious Polynomial Evaluation (OPE) [37,38], a protocol where a sender with a polynomial $P(x)$ and a receiver with input α interact so the receiver learns $P(\alpha)$, while the sender learns nothing about α [12,26,27,33]. Receiver privacy[4] [38] does not stop a malicious sender from using a higher-degree polynomial. Therefore, if an adversary wants to extract P, he needs an adaptive strategy to guess the polynomial's degree. OPE serves as a building block for numerous cryptographic protocols,

[4] The sender cannot distinguish between two inputs sent by the receiver.

including RSA key generation [24], oblivious keyword search [19] and electronic voting systems [40]. Additionally, OPE plays a key role in Privacy-Preserving Machine Learning [14] (a typical use case is in healthcare[5]).

Structure of the Paper. In Section 2, we provide the necessary mathematical background regarding the problem under consideration. Our first approach to polynomial evaluation is presented in Section 3, followed by an alternative method in Section 4. Implementation results are given in Section 5. Section 6 introduces an aggregation problem. We conclude and provide the reader with future research directions in Section 7.

Notations. Throughout the paper, the notation $\#S$ denotes the cardinality of a set S. The assignment of value y to variable x is denoted by $x \leftarrow y$. The subset $\{0, \ldots, s\} \in \mathbb{N}$ is denoted by $[0, s]$. A vector v of length n is denoted either $v = (v_0, \ldots, v_{n-1})$ or $v = \{v_i\}_{i \in [0,n-1]}$.

2 Preliminaries

2.1 Recursively Constructing Polynomials Based on Additions and Multiplications

Let $a_1, \ldots, a_m$ be m symbolic variables. We define the sets of arithmetic expressions G_k obtained by combining a_1 to a_m as follows

$$\begin{aligned} G_0 &= \{a_1, \ldots, a_m\} \\ &\ldots \\ G_k &= \{a + b, a, b \in G_{k-1}\} \bigcup \{a \cdot b, a, b \in G_{k-1}\} \end{aligned} \tag{1}$$

We further provide an example for $m = 3$. An automated construction of such sets yields

$$\begin{aligned} G_1 &= \{2a_1, a_1 + a_2, a_1 + a_3, 2a_2, a_2 + a_3, 2a_3, a_1^2, a_1a_2, a_1a_3, a_2^2, a_2a_3, a_3^2\} \\ \#G_2 &= 136 \\ \#G_3 &= 15203 \\ &\ldots \end{aligned}$$

Ignoring collisions we can derive an upper bound for $\#G_k$. More precisely, starting from G_{k-1} and using additions we can construct $\#G_{k-1}(\#G_{k-1}+1)/2$ new elements. The same number of elements are obtained using multiplication. Therefore, we have

$$\#G_k \leq \#G_{k-1}(\#G_{k-1} + 1).$$

[5] where a patient seeks a health status prediction from a company that holds a trained model, without disclosing any sensitive personal health data [25]

We define the following recurrence

$$V_1 = \#G_0 = m,$$
$$V_k = V_{k-1}(V_{k-1} + 1).$$

Using the methods developed in [1,3], we can derive a tight upper bound for V_k given the value of m. The first step in obtaining this bound is to compute V_0. More precisely, V_0 is defined as the positive root of the quadratic equation $x^2 + x - m = 0$.

We observe that the following series converges rapidly

$$\sum_{k\geq 0} \frac{1}{2^{k+1}} \cdot \log\left(1 + \frac{1}{V_k}\right) \to \alpha.$$

It can then be shown that $V_k \leq \theta^{2^k} - 1/2$, where $\theta = V_0 \cdot \exp(\alpha)$. Consequently, we obtain

$$\#G_k \leq \theta^{2^{k+1}}.$$

For completeness, we provide some examples for θ in Table 1. Note that the value for $m = 2$ was previously computed by Knuth in [1].

m	2	3	4	5	6	7	8	9	10
V_0	1	1.30277	1.56155	1.79128	2	2.19258	2.37228	2.54138	2.70156
θ	1.59791	1.88067	2.12799	2.35011	2.55331	2.74167	2.91801	3.08435	3.24221

Table 1. Numerical approximations for θ.

The next properties we want to determine are the upper bounds for the coefficients and the degrees of the monomials of polynomials in G_k. We begin with an illustrative example by considering the case $m = 3$. Starting with a_1, a_2, a_3 and, for instance, adding and multiplying a_1 by itself, we obtain $2a_1$ and a_1^2. Thus, we obtain the largest coefficient and the largest power in G_1, namely 2. If we multiply $2a_1$ and a_1^2 by themselves, we get $4a_1^2$ and a_1^4, yielding the value 4 as an upper bound for both the coefficients and the degree. We continue this process in Table 2, where a pattern begins to emerge.

k		
1	$2a_1$	a_1^2
2	$4a_1^2 = 2^{2^1} a_1^{2^1}$	$a_1^4 = a_1^{2^2}$
3	$16a_1^4 = 2^{2^2} a_1^{2^2}$	$a_1^8 = a_1^{2^3}$
4	$256a_1^8 = 2^{2^3} a_1^{2^3}$	$a_1^{16} = a_1^{2^4}$
5	$65536a_1^{16} = 2^{2^4} a_1^{2^4}$	$a_1^{32} = a_1^{2^5}$

Table 2. The maximum coefficient and degree in G_k for $k \leq 5$.

We further provide formal upper bounds for the coefficients and the degrees of the monomials of polynomials in G_k.

Lemma 1. *Let $k \geq 1$. If*

$$P(a_1, \ldots, a_m) = \sum_{i_1,\ldots,i_m} c_{i_1,\ldots,i_m} a_1^{u^{(1)}_{i_1,\ldots,i_m}} \ldots a_m^{u^{(m)}_{i_1,\ldots,i_m}} \in G_k,$$

then

$$u^{(1)}_{i_1,\ldots,i_m} + \ldots + u^{(m)}_{i_1,\ldots,i_m} \leq 2^k \quad \textit{and} \quad c_{i_1,\ldots,i_m} \leq 2^{2^{k-1}}.$$

Proof. We will prove this lemma using induction. We define $\max_k(u^{(1)}_{i_1,\ldots,i_m} + \ldots + u^{(m)}_{i_1,\ldots,i_m})$ as the maximum degree of monomials in $a_1, \ldots, a_m$ for any $P(a_1, \ldots, a_m) \in G_k$. Also, let $\max_k(c_{i_1,\ldots,i_m})$ be the largest coefficient of any $P(a_1, \ldots, a_m)$.

When $k = 1$, it is easy to see that $\max_1(u^{(1)}_{i_1,\ldots,i_m} + \ldots + u^{(m)}_{i_1,\ldots,i_m}) = 2^1$ and $\max_1(c_{i_1,\ldots,i_m}) = 2^{2^0}$. When $k = 2$, we obtain that $\max_2(u^{(1)}_{i_1,\ldots,i_m} + \ldots + u^{(m)}_{i_1,\ldots,i_m}) = 2^2$ and $\max_2(c_{i_1,\ldots,i_m}) = 2^{2^1}$.

We assume that the lemma is true for k and we prove it for $k + 1$. The only strategy that maximizes the degree of a monomial from G_{k+1} is to choose a maximal monomial from G_k and multiply it by itself. For example, we can choose $a_1^{2^k} \in G_k$. Therefore, we have $\max_{k+1}(u^{(1)}_{i_1,\ldots,i_m} + \ldots + u^{(m)}_{i_1,\ldots,i_m}) = 2^k + 2^k = 2^{k+1}$. We can also see that multiplying $2^{2^{k-1}} a^{2^{k-1}}$ to itself leads to $\max_{k+1}(c_{i_1,\ldots,i_m}) = 2^{2^{k-1}} \cdot 2^{2^{k-1}} = 2^{2^k}$. □

2.2 Defining the Polynomial Inference Problem

In this subsection, we present a high-level description of the protocol used to infer the polynomial $P(x_1, \ldots, x_m)$. More precisely, for a given $k \in \mathbb{N}$, Alice (the malicious user) and Bob (the victim) exchange information that enables Alice, without Bob's consent, to compute the polynomial P:

1. Bob selects a polynomial $P \in G_k$.
2. Alice chooses m integers $a_1, \ldots, a_m \in \mathbb{N}^*$ and sends them to Bob.
3. Bob computes $r = P(a_1, \ldots, a_m)$ using the values received from Alice and then sends the result back to her.
4. Given r, Alice attempts to infer the polynomial P.
5. If Alice is not successful, she repeats steps 2 and 3 until she collects enough data to determine P.

A graphical representation of the overall process is given in Figure 1. The objective of this paper is to develop a series of algorithms for inferring P in the aforementioned scenario.

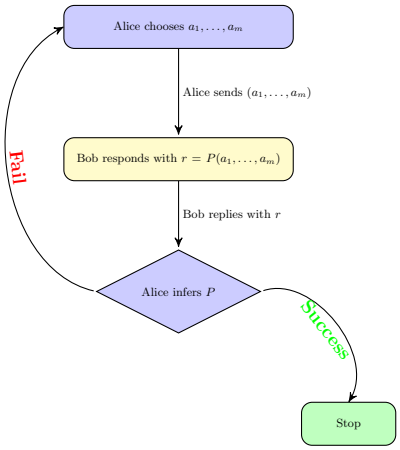

Fig. 1. The overall process.

3 Interpolating Multivariate Polynomials

To illustrate the underlying idea, we begin with the following example. Let

$$P(x, y, z, t) = x^3t + x^2 + xy + xz + y^2z + yt^2 + zt.$$

Alice has access to nine values of the form $P_i = P(i + 1, a_2, a_3, a_4)$ for $i \in [0, 8]$. To reduce the problem of recovering P to the recovery of three polynomials in x, y, and z, Alice proceeds as follows:

1. Compute the values $P_{i,t^0} = P(i + 1, a_2, a_3, a_4) \bmod a_4$. This corresponds to recovering the polynomial $P_{t^0}(x, y, z) = x^2 + xy + xz + y^2z$ from the points P_{i,t^0}.
2. Once $P_{t^0}(x, y, z)$ is recovered, compute

   $$\tilde{P}_i = \frac{P_i - P_{t^0}(i + 1, a_2, a_3)}{a_4}.$$

   This corresponds to working with the polynomial

   $$\frac{P(x, y, z, t) - P_{t^0}(x, y, z)}{t} = x^3 + yt + z.$$

   Hence, in this step Alice recovers the polynomial $P_{t^1}(x, y, z) = x^3 + z$ from the values $P_{i,t^1} \equiv \tilde{P}_i \bmod a_4$.
3. Finally, Alice computes

   $$\hat{P}_i = \frac{\tilde{P}_i - P_{t^1}(i + 1, a_2, a_3)}{a_4}.$$

   Hence, she can then recover the polynomial $P_{t^2}(x, y, z) = y$ from the values $P_{i,t^2} \equiv \hat{P}_i \bmod a_4$.

Algorithm 1: *Interpolate*$_1$

Input: $n+1$ positive integer pairs $\{a_{1,i}, r_i\}_{i\in[0,n]}$.
Output: A polynomial $P(x_1)$ such that $r_i = P(a_{1,i})$.

1 Compute using Lagrange interpolation

$$P(x_1) = \sum_{i=0}^{n} r_i \cdot \ell_i(x_1) \bmod a_2,$$

where

$$\ell_i(x_1) = \prod_{\substack{t=0 \\ t\neq i}}^{n} \frac{x_1 - a_{1,t}}{a_{1,i} - a_{1,t}} \bmod a_2.$$

2 **return** $P(x_1)$

This method can be easily adapted to reduce the recovery of polynomials in x, y, and z to successive recoveries in fewer variables, first in x and y, and ultimately in x alone. Note that recovering polynomials modulo a_2 in one variable can be achieved using Lagrange's interpolation method. Our proposed method is fully described in Algorithm 2. An important and necessary component in resolving the problem stated in Section 2.2, which addresses the realistic scenario in which the attacker has no prior knowledge of the degree of the target polynomial, is provided in Algorithm 3. We further present a detailed analysis of the formal algorithms and how Alice can address the issues that arise.

Failure points. Our method has several potential failure points that must be addressed. The first issue arises for polynomials in two variables. If a_2 is smaller than the polynomial coefficients, then the recovered polynomial may differ from the original one. For example, consider the following polynomial held by Bob

$$P(x, y) = 16x^2y^5 + x^2y + 5xy^2 + y^4 + y^3.$$

If Alice sets $a_{1,i} = i+1$ for $i \in [0, 8]$ and $a_2 = 11$, using our method she recovers

$$P'(x, y) = x^2y^6 + 5x^2y^5 + x^2y + 5xy^2 + y^4 + y^3.$$

instead of the original $P(x, y)$. Note that we used as input points Hence, she must ensure that $a_2 \geq 17$.

A second issue arises from reducing values modulo multiple moduli. For $m > 2$, let

$$P(x_1, \ldots, x_m) = P_0(x_1, \ldots, x_{m-1}) + \ldots + P_\alpha(x_1, \ldots, x_{m-1})x_m^\alpha. \tag{2}$$

If we do not ensure that $a_m > P_j(a_{1,i}, a_2, \ldots, a_{m-1})$ for $j \in [0, \alpha]$, then reducing modulo a_m we obtain values smaller than the actual ones. One way to ensure this is by having a_3 bigger than all the coefficients of $P(a_{1,i}, a_2, x_3, \ldots, x_m)$,

Algorithm 2: $Interpolate_m$, when $m \geq 2$

Input: Primes $\{a_i\}_{i\in[2,m]}$ and $n+1$ positive integer pairs $\{a_{1,i}, r_i\}_{i\in[0,n]}$.
Output: A polynomial $P(x_1, \ldots, x_m)$ such that $r_i = P(a_{1,i}, a_2, \ldots, a_m)$.

1 $j \leftarrow 0$
2 $\{R_i\}_{i\in[0,n]} \leftarrow \{r_i\}_{i\in[0,n]}$
3 **while** 1 **do**
4 Compute $\{S_i\}_{i\in[0,n]} \leftarrow \{R_i \bmod a_m\}_{i\in[0,n]}$.
5 Call the function $L_j \leftarrow Interpolate_{m-1}(\{a_i\}_{i\in[2,m-1]}, \{a_{1,i}, S_i\}_{i\in[0,n]})$.
6 **if** $L_j = \perp$ **then**
7 **return** $\perp$
8 Compute $\{R_i\}_{i\in[0,n]} \leftarrow \{R_i - L_j(R_i, a_2, \ldots, a_{m-1})\}_{i\in[0,n]}$.
9 **if** *all* $R_i = 0$ **then**
10 **break**
11 **if** *any* $R_i < 0$ **then**
12 **return** $\perp$
13 Compute the largest d_j such that $(a_m)^{d_j}$ divides all R_i
14 Compute $\{R_i\}_{i\in[0,n]} \leftarrow \{R_i/(a_m)^{d_j}\}_{i\in[0,n]}$
15 $j \leftarrow j+1$
16 $P(x_1, \ldots, x_m) \leftarrow L_j(x_1, \ldots, x_{m-1})$
17 **for** $t \leftarrow j-1$ **downto** 0 **do**
18 Compute the polynomial

$$P(x_1, \ldots, x_m) \leftarrow P(x_1, \ldots, x_m) \cdot (x_m)^{d_t} + L_t(x_1, \ldots, x_{m-1})$$

19 **return** $P(x_1, \ldots, x_m)$

a_4 bigger than all the coefficients of $P(a_{1,i}, a_2, a_3, x_4 \ldots, x_m)$ and so on. For example, consider Bob's polynomial

$$P(x, y, z) = x^2 + xyz + xy + y^3z + z^2.$$

If Alice runs $Interpolate_3$ with variables x, y, z and primes $a_2 = 13$ and $a_3 = 2131$, she obtains

$$P'(x, y, z) = x^2 + xyz + xy + 5yz + 2z^2 + z.$$

since we have $2210 \leq z_1 \leq 2314$, where $P(i+1, 13, z) = z^2 + z_1 z + z_0$ for $i \in [0, 8]$. To recover Bob's P, Alice can adopt two strategies: either set $a_2 = 2333$, or permute the variables (*i.e.* consider P as $\tilde{P} = x^2 + xzy + xz + z^3y + y^2$) and adapt the algorithm accordingly. Note that permuting the variables yields $170 \leq \tilde{z}_0 \leq 250$ and $14 \leq \tilde{z}_1 \leq 126$, where $\tilde{P}(i+1, 13, z) = \tilde{z}^2 + \tilde{z}_1 z + \tilde{z}_0$ for $i \in [0, 8]$. Therefore, the issue raised above is resolved. Moreover, by examining the coefficients z_0 and z_1, we observe that Alice can further reduce the value of a_2 to 251.

Another issue may occur in the while loop of Algorithm 2 on line 13. For example, when computing the largest exponent d_j of a_m, it may happen that

Algorithm 3: Probes Bob until it finds the correct P.

Input: Primes $\{a_i\}_{i\in[2,m]}$ such that $a_2 < a_3 < \ldots < a_m$.
Output: A polynomial $P(x_1, \ldots, x_m)$.

1 $\mathcal{L} \leftarrow \emptyset$, $j \leftarrow 3$
2 Interrogate Bob on points $\{(i+1, a_2, \ldots, a_m)\}_{i\in[0,j]}$ and receive $\{r_i\}_{i\in[0,j]}$
3 Use Algorithm 2 with input $(\{a_i\}_{i\in[2,m]}, \{i+1, r_i\}_{i\in[0,j-1]})$ and receive an answer P
4 **if** $P \neq \perp$ *and* $P(j+1, a_2, \ldots, a_m) = r_j$ **then**
5 | **return** P
6 **while** 1 **do**
7 | **if** $j > a_2$ **then**
8 | | **return** $\perp$
9 | $j \leftarrow j+1$
10 | Interrogate Bob on point $(j+1, a_2, \ldots, a_m)$ and receive r_j
11 | Use Algorithm 2 with input $(\{a_i\}_{i\in[2,m]}, \{i+1, r_i\}_{i\in[0,j-1]})$ and receive an answer P
12 | **if** $P \neq \perp$ *and* $P(j+1, a_2, \ldots, a_m) = r_j$ **then**
13 | | **return** P

$a_m \mid P_1(a_{1,i}, a_2, \ldots, a_{m-1})$ for all i (see Equation (2)). In this case, the computed power will be larger than the correct one. This issue can be mitigated by choosing all a_i to be prime numbers. Then the probability of obtaining the correct power in a single iteration is approximately $1 - 1/a_i$, which is non-negligible when a_i is sufficiently large.

Finally, a failure may occur if insufficient input data are used. In such a case, some R_i values (see Algorithm 2) may become negative, and the algorithm returns $\perp$, indicating that the computed polynomial is incorrect. Since Alice does not know the exact degree of P, she uses Algorithm 3 to recover the correct polynomial and thereby avoids this issue. More precisely, Alice queries Bob until Algorithm 2 returns a polynomial that maps j points to r_j and also satisfies the additional condition $P(j+1, a_2, \ldots, a_m) = r_j$. This condition prevents the case where insufficient data are used and all R_i values become 0. Note that on line 7 an additional check is performed to avoid the situation where a_2 divides one of the coefficients of P. Specifically, if $j = a + k$, then $j - k$ will not be invertible when computing the $\ell_j(x_1)$ polynomial. If Algorithm 3 fails, Alice must increase the a_i values or permute the variables.

A second approach. Another possible approach to solving the problem is to reduce modulo $a_t, \ldots, a_m$ and apply a $(t-1)$-variate polynomial interpolation technique. When $t = 2$, this yields our proposed method, while for $t = m+1$, it reduces to the classical multivariate interpolation (see [43]). We do not pursue this trade-off, as our primary objective is to minimize the number of queries to Bob. Note that, for $t > 2$ the number of required points increases to $\binom{\deg(P)+t}{t}$, where $\deg(P)$ denotes the total degree of the polynomial P.

4 Lattice-Based MKP Resolution Approaches for Reconstructing Multivariate Polynomials

4.1 Preliminaries

We further provide the reader with the necessary mathematical background for solving the problem presented in Section 2.2 by means of different methods than in Section 3. The key ingredient of our alternative approach is solving the *modular knapsack problem* (MKP).

4.1.1 Modular Knapsack Problems

The subset sum problem (SSP) [20] and the knapsack problem [35] are two very important NP-complete computational problems in computer science, with various applications in cryptography. The subset sum problem aims at finding a subset of a given set of integers whose sum equals a given target value, while the knapsack problem involves selecting a subset of items with maximum value while adhering to a weight constraint.

There are some notable differences between the two problems. The aim of the subset-sum problem is to find a subset that adds up to a specific value, while the knapsack problem implies maximizing the value of a subset subject to a weight constraint.

As already underlined, our central interest in this section is MKP, an important variation of the classical knapsack problem. In this case, the items are represented by a value, a weight, and a modular coefficient, and the scope is to obtain a subset of items that maximizes the total value while respecting a weight constraint and a modular constraint.

Definition 1 (The modular knapsack problem). *Let $a_1, \ldots, a_n$ be positive integers and $S, M \in \mathbb{Z}$. The generic modular knapsack problem consists of finding $e_1, \ldots, e_n \in \mathbb{N}$ such that*

$$\sum_{i=1}^{n} a_i e_i \equiv S \pmod{M}. \tag{3}$$

Remark 1. The $0-1$ modular knapsack problem differs from Definition 1 by the fact that $e_i \in \{0, 1\}$ for $i \in [1, n]$.

In this paper, our main interest is in the standard modular knapsack problem.

Remark 2 (Equivalence between SSP and MKP). Under some assumptions, we have an equivalence between solving the classical subset sum problem and the modular subset sum problem. When we are not dealing with polynomial factors, basically, any algorithm that solves one of the problems can be used to find solutions for the other.

Remark 3 (A generic resolution algorithm for MKP). Given an algorithm that solves a knapsack problem over the integers we have the following:

1. Consider Equation (3) with $a_i \in [0, M-1]$;
2. It follows immediately that any sum of at most n numbers is in $[0, nM-1]$ (*e.g.* a_i);
3. If $S \in [0, M-1]$, then solving n knapsacks over the integers with target sums $S, S+M, \ldots, S+(n-1)M$ means solving the modular knapsack given by Equation (3).

Definition 2 (The Density of Subset Sum Algorithms.). *The density of a set* $\{a_1, ..., a_n\}$ *of weights is defined as*

$$d = \frac{n}{\log_2 \max(a_i)}. \tag{4}$$

Definition 3 (The Density of Generic MKP). *Consider the general modular knapsack problem*

$$\sum_{i=1}^{n} a_i e_i \equiv S (\mathrm{mod} M),$$

where $a_1, \ldots, a_n$ *are positive integers* $S, M \in \mathbb{Z}$ *and* $e_1, \ldots, e_n \in \{0, 1, \ldots, u-1\}$. *We define the density as*

$$d = \frac{n \cdot \log_2 u}{\log_2 \max(a_i, M)}$$

Remark 4. In order to solve low-density knapsacks, lattice reduction is a very useful tool. According to [5], lattice reduction-based solutions are not an option when the density of the knapsack is close to 1.

4.1.2 Lattice Reduction: a Tool for Solving Modular Knapsacks

We refer the reader to [28] for basic definitions and properties of lattices as these concepts exceed the scope of our paper.

Low-density knapsacks. A notable method for solving low-density instances of MKP was presented in [31] as the Lagarias–Odlyzko algorithm (LO), relying on a shortest vector problem (SVP) oracle. We recall LO in Appendix A (Algorithm 7). For random knapsack problems, LO can solve instances with density $d < 0.64$, given an oracle solving SVP.

LO was further improved in [13] through modifications of the underlying lattice construction. We further refer to the attack in [13] as CJLOSS. This algorithm works for densities up to $d < 0.94$, by means of a simple modification detailed in Appendix A.

Remark 5. The shortest vector oracle is replaced in practice by a lattice reduction algorithm [5], such as LLL or BKZ [44].

Hard knapsacks. An algorithm for solving knapsacks with density almost 1 (also known as hard knapsacks) was presented in [45]. Being a generic solution (based on the birthday paradox) is, naturally, less practical than low-density attacks like LO and CJLOSS. Building upon the results in [45], more practical algorithms for hard knapsacks were discussed in [29]. The techniques developed before were extended in [5].

Lattice reduction speed-ups. On the practical side, LLL-inspired algorithms and their implementations were further discussed in the literature. Until recently, L^2 was considered the state-of-the-art lattice reduction algorithm [39], being implemented in fpLLL [2]. Various research directions regarding LLL-type algorithms continued to emerge. A polynomial-time LLL variant inspired by Lagrange's notion of pairwise reduction was introduced in [4]. The method is characterized by a significant gain in the direction of precision over speed, producing basis vectors with the norm of first vector up to 16% shorter than the one obtained with classical LLL. Also, it improves BKZ preprocessing quality by approximately 3%. A further noteworthy result, though somewhat tangential to the present discussion, is presented in [41], where the authors derive sharper complexity estimates for discrete Gaussian sampling over BKZ-reduced bases.

4.2 Main Algorithm

We consider the equation

$$r = \sum_{i=1}^{n} c_i \cdot a_1^i \bmod a_2 \bmod a_3 \ldots \bmod a_m \tag{5}$$

and proceed with the modular reduction recursively, in a reverse manner. We apply a knapsack-based solution algorithm at each step to recover the coefficients.

Algorithm 4: $Knapsack_1$

Input: A prime a_2 and a positive integer pair $\{a_1, r\}$.
Output: A bivariate polynomial $P(x)$ such that $r = P(a_1)$.

1 Solve

$$r = \sum_{i=1}^{n} t_i \cdot a_1^i \mod a_2 \text{ for } \{t_i\}$$

using a modular knapsack resolution algorithm

2 $P(x) \leftarrow \sum_{i=1}^{n} t_i x^i$

3 **return** $P(x)$

If, in addition, we restrict k to a value such that $a_i > 2^{2^{k-1}}$, for $i \in [2, m]$, and $a_2 < \ldots < a_m$, then we are assured that the values found by solving the modular

knapsack are also valid in $\mathbb{Z}$. Hence, the integer values can be subtracted from r to reveal a polynomial that can be divided by a proper power of a_i before applying modular reduction iteratively in the case of Equation (5). When we reach 0, the algorithm is run backward to reconstruct the polynomial P. The method is fully described in Algorithm 5.

An important restriction from Section 3 that is also valid in this context is the fact that a_m must be larger than $P_j(a_1, \ldots, a_{m-1})$ for $j \in [0, \alpha]$, where

$$P(x_1, \ldots, x_m) = \sum_{j=0}^{\alpha} P_j(x_1, \ldots, x_{m-1}) x_m^j$$

This issue appears from reducing values modulo multiple moduli. To illustrate the importance of this issue, we consider that Alice sets the values $a_1 = 11$, $a_2 = 139$ and $a_3 = 1669$ and Bob selects the following polynomial

$$P(x, y, z) = 2x + xy + yz + xyz + z^2.$$

In this case the correct polynomial is recovered by Alice from the point $r = 5571004$ using $Knapsack_m$. If Alice replaces the value of a_3 with 1667 the algorithm fails since $2a + ab = 1551$ and $b + ab = 1668$.

Algorithm 5: $Knapsack_m$

Input: A positive integer pair $\{a_1, r\}$ and prime integers $a_2 < a_3 < \ldots < a_m$.
Output: A multivariate polynomial P such that $r = P(a_1, \ldots, a_m)$.

if $r \bmod a_m = 0$ **then**
  Compute the largest $d_k^m \in \mathbb{N}$ such that $a_m^{d_k^m}$ divides r
  Compute $r' \leftarrow r / a_m^{d_k^m}$
$j \leftarrow 0$
$r_0 \leftarrow r'$
while $r_j \neq 0$ **do**
  Compute $s \leftarrow r_j \bmod a_m$
  Call the function $r'_j \leftarrow Knapsack_{m-1}((a_1, s), a_2, \ldots, a_{m-1})$
  Compute $r_{j+1} \leftarrow r_j - r'_j(a_1, \ldots, a_{m-1})$
  Compute the largest $d_j \in \mathbb{N}$ such that $a_m^{d_j}$ divides r_{j+1}
  Compute $r_{j+1} \leftarrow r_{j+1} / a_m^{d_j}$
  $j \leftarrow j + 1$
$P(x_1, \ldots, x_m) \leftarrow 1$
for $\ell \leftarrow j - 1$ **downto** 0 **do**
  Compute the polynomial

$$P(x_1, \ldots, x_m) \leftarrow P(x_1, \ldots, x_m) \cdot x_m^{d_\ell} + r'_j(x_1, \ldots, x_{m-1})$$

$P(x_1, \ldots, x_m) \leftarrow x_m^{d_k^m} \cdot P(x_1, \ldots, x_m)$
return $P(x_1, \ldots, x_m)$

Remark 6 (Success probability). Note that for some particular instances, our Algorithm 5 falls into the hard knapsacks category. Hence, the probability of success of our proposed solution is directly linked to the selected modular knapsack resolution algorithm.

To clarify further, we consider hard instances the ones being characterized by a density λ of the knapsack, where $0.94 \leq \lambda < 1$. Such instances are not a case of interest for the current work, given that the probability of such an event to happen is negligible for the sets G_k (even 0 for the examples we consider).

As already discussed, lattice reduction-based algorithms are suitable in the low-density case, *e.g.* smaller than λ, where $\lambda = 0.94$ in the CJLOSS case). When $0.94 \leq \lambda < 1$, we stress that the algorithms in [29] can be considered suitable options.

Assuming that the selected modular knapsack algorithm succeeds with probability $\mathcal{P}$ and taking into account the recursive structure of our proposed method, we derive an analysis for $m = 3$, which can easily be generalized to higher values.

Let $P(x_1, x_2, x_3) \in G_k$, for some k, and consider

$$P(x_1, x_2, x_3) = P_0(x_1, x_2) + P_1(x_1, x_2)x_3 + \ldots + P_n(x_1, x_2)x_3^n.$$

Let us assume that $P_0(x_1, x_2)$ can be found with probability $\mathcal{P}$.

Given this, we can state that $P_1(x_1, x_2)$ can only be found using the value of P_0 and the formula $\frac{P-P_0}{x_3}$.

Thus, the probability of identifying P_1 is $\mathcal{P}^2$. Extrapolating, we conjecture that the probability of successfully determining the polynomial P_n becomes $\mathcal{P}^{n+1}$.

Remark 7 (Parameter selection). We examine MKP under the assumption that $d \geq \lambda$, for some λ. It can be easily observed that the density can be computed through an iterative procedure, with the core analysis reducing to the case of two variables. It follows that

$$\frac{2^k}{\log_2 \max(a_1^i, a_2)} \geq \lambda.$$

We suppose that the modulus a_2 is smaller than the highest power of a_1, since the complementary case is negligible. Thus, we have that

$$2^k \geq \lambda \cdot \log_2 a_1^{\deg_{a_1} P} \Leftrightarrow 2^k \geq \lambda \cdot \deg_{a_1} P \cdot \log_2 a_1 \Leftrightarrow$$
$$\log_2 a_1 \leq \frac{2^k}{\lambda \cdot \deg_{a_1} P}$$

It follows that $d \geq \lambda$ if and only if the number of bits in a_1 is less than $\frac{2^k}{\lambda \cdot \deg_{a_1} P}$.

A second approach. An alternative way to solve the problem is to reduce modulo some $a_t, \ldots, a_m$ and apply a $(t-1)$-modular knapsack resolution algorithm. Note that when $t = 2$, this coincides with our proposed method. This approach is out of the scope of our current paper, and thus we leave it as an open problem.

5 Implementation and Optimal Execution Recommendations

To validate our hypotheses and algorithms, we implemented a set of reference (unoptimized) versions. Our Lagrange interpolation-based Algorithm 2 was implemented in Python, while the lattice reduction-based Algorithm 5 was implemented using Mathematica 14.3 online and uses the resolution algorithm provided in [5].

5.1 Performance Analysis

Table 3 summarizes both the number of queries required to reconstruct the polynomial P and the associated computational complexity. We further denote by $deg(P)$ the highest degree of P and by $deg_{a_i}(P)$ the degree of P in the variable a_i. For convenience, we define $\rho(m) = \binom{deg(P)+m}{m}$.

If Alice uses classical multivariate interpolation, she requires one additional query compared to the number of queries given in [30,43] (*i.e* $\rho(m)$), because an additional evaluation point is necessary to verify the correctness of the recovered polynomial P. This extra query ensures that the actual degree of P does not exceed the conjectured degree.

The method presented in Algorithm 3 requires $deg_b(P)+1$ evaluation points to reconstruct P, along with one further point for verification, assuming that a_2 exceeds all coefficients of P and that $a_k > P_j(a_{1,i}, a_2, \ldots, a_{k-1})$ for every $k \in [3, m]$. When using our lattice based method (Algorithm 5), Alice needs only a single evaluation point to determine P, plus an additional point to confirm correctness.

For the complexity analysis of Algorithm 2, we use that Lagrange interpolation runs in $\mathcal{O}(deg_{a_1}(P)^2)$ time (see [42]), while naive polynomial evaluation has complexity $\mathcal{O}(deg(P)^2)$. In the case of the knapsack-based method Algorithm 5, we write $O = \mathcal{O}(\mathcal{A})$, where $\mathcal{A}$ denotes a modular knapsack solving procedure.

Our Lagrange interpolation-inspired algorithm is deterministic and requires a larger number of queries to ensure a correct result. In contrast, our method relying on modular knapsack resolution algorithms follows a probabilistic approach, but requires only 2 queries to reconstruct the polynomial P.

Table 3. Performance analysis for inferring P

Algorithm	Number of Querries	Complexity
Classical multivariate interpolation [30,43]	$\rho(m)+1$	$\mathcal{O}(\rho(m)^2 + deg(P)^2)$
Algorithm 3	$deg_{a_1}(P)+2$	$\mathcal{O}\left(deg_{a_1}(P)\left(deg^2_{a_1}(P) deg_{a_2}(P)\ldots deg_{a_m}(P) + deg(P)^2\right)\right)$
Algorithm 5	2	$\mathcal{O}\left(O \cdot deg_{a_2}(P)\ldots deg_{a_m}(P) + deg(P)^2\right)$

5.2 Optimal Execution Recommendations

Given the results presented in Table 3, taking advantage both of the reduced number of queries needed in Algorithm 5 and of the deterministic nature of Algorithm 3, directly guides us towards an optimal strategy (in terms of success probability maximization) for running our proposed methods. Therefore, to minimize the number of queries sent to Bob, we suggest initially applying Algorithm 5 and resorting to Algorithm 3 only if the first approach is unsuccessful. In the event that an incorrect polynomial is recovered, Alice may proceed in one of the next three ways:

1. Modify the a_i values[6] and repeat the procedure;
2. Permute variables[7] and repeat the procedure;
3. Switch to a classical multivariate interpolation method, *e.g.* [43].

A schematic overview of this workflow is illustrated in Figure 2, where CMI refers to a classical multivariate interpolation algorithm.

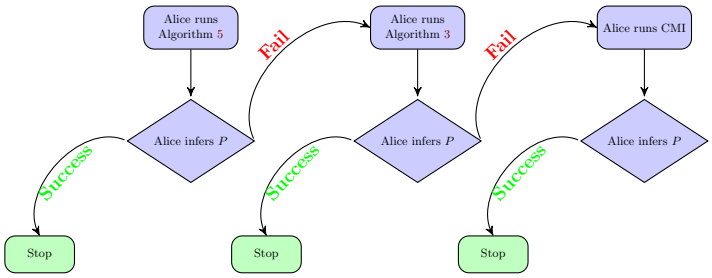

Fig. 2. The recommended process.

6 Aggregating Polynomials

6.1 Defining the Problem

In this subsection, we present a high-level description of a data aggregation problem. More precisely, we assume that we have a series of α servers S_i that get as input m_i variables $a_{i,1}$ to a_{i,m_i} and return $P_i(a_{i,1}, \ldots, a_{i,m_i})$, where $i \in [1, \alpha]$. At some point in time the servers are integrated into an $\alpha + 1$ server $S_{\alpha+1}$

[6] In order to ensure that the a_i values exceed the polynomial coefficients and that the resulting polynomials remain in $\mathbb{Z}[x_1, \ldots, x_{m-1}]$.
[7] In order to ensure that the resulting polynomials remain in $\mathbb{Z}[x_1, \ldots, x_{m-1}]$.

that takes as input the polynomial evaluations of the α servers and returns the polynomial evaluation of

$$R(\bar{a}_{1,1}, \ldots, \bar{a}_{1,m_1}, \ldots, \bar{a}_{\alpha,1}, \ldots, \bar{a}_{\alpha,m_\alpha}) \\ = Q(P_1(\bar{a}_{1,1}, \ldots, \bar{a}_{1,m_1}), \ldots, P_\alpha(\bar{a}_{\alpha,1}, \ldots, \bar{a}_{\alpha,m_\alpha})).$$

A graphical representation of the aggregation process is given in Figure 3.

We can describe the interaction between Alice and the servers in two phases:

Phase 1: Interactions with the α servers S_i

1. Alice chooses m_i integers $a_{i,1}, \ldots, a_{i,m_i} \in \mathbb{N}^*$ and sends them to the ith server.
2. The server S_i computes $r_i = P_i(a_{i,1}, \ldots, a_{i,m_i})$ using the values received from Alice and then sends the result back to her.
3. Given r_i, Alice attempts to infer the polynomial P_i.
4. If Alice is not successful, she repeats steps 2 and 3 until she collects enough data to determine P_i.

Phase 2: Interaction with the $\alpha + 1$ server $S_{\alpha+1}$

1. Alice chooses $m_1 + \ldots + m_\alpha$ integers $\bar{a}_{i,j_i} \in \mathbb{N}^*$ for $i \in [1, \alpha]$ and $j \in [1, m_i]$, and sends them to the server.
2. The server $S_{\alpha+1}$ computes $r = R(\bar{a}_{1,1}, \ldots, \bar{a}_{1,m_1}, \ldots, \bar{a}_{\alpha,1}, \ldots, \bar{a}_{\alpha,m_\alpha})$ using the values received from Alice and then sends the result back to her.
3. Given r, Alice attempts to infer the polynomial R.
4. If Alice is not successful, she repeats steps 2 and 3 until she collects enough data to determine R.

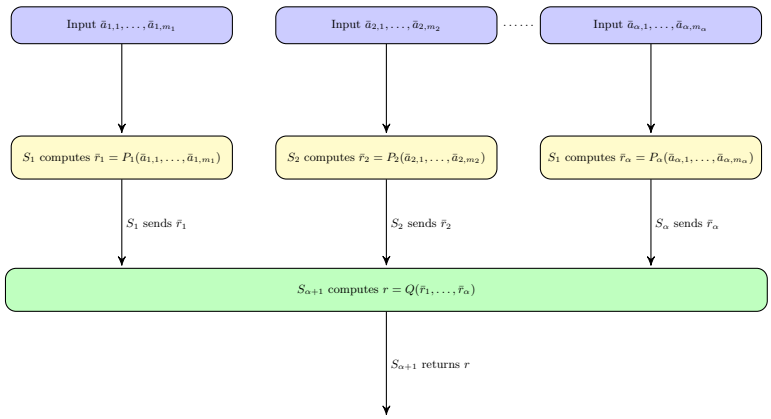

Fig. 3. The aggregation process.

6.2 A Possible Solution

A straightforward approach for recovering R is to use either Algorithm 3 or Algorithm 5. This amounts to reconstructing a polynomial in $m_1 + \ldots + m_\alpha$ variables, where both the number of required points and the computational complexity depend on $\deg_{a_i}(R)$.

In Algorithm 6, we propose an alternative approach. For simplicity, we describe it using Algorithm 5, although Algorithm 3 can be employed as well. The key advantage of our method is that it recovers polynomials with a smaller number of variables, thereby reducing both the number of required points and the overall computational complexity.

Algorithm 6: Tries to compute the polynomial R.

Phase 1:

for $i \in [1, \alpha]$ **do**
  Input: Prime integers $a_{i,1} < a_{i,2} < a_{i,3} < \ldots < a_{i,m_i}$.
  Output: A multivariate polynomial P_i.
  Interrogate S_i with $a_{i,1}, \ldots, a_{i,m}$ to obtain r_i.
  Use $Knapsack_{m_i}((a_{i,1}, r_i), a_{i,2}, \ldots, a_{i,m_i})$ to recover polynomial P_i.
return $P_1, \ldots, P_\alpha$

Phase 2:

Input: The multivariate polynomials $P_1, \ldots, P_\alpha$.
Output: A multivariate polynomial R.
for $i \in [1, \alpha]$ *and* $j \in [1, m_i]$ **do**
  Choose $\bar{a}_{i,j}$ until $\beta_i = P_i(\bar{a}_{i,1}, \ldots, \bar{a}_{i,m})$ are prime and $\beta_1 < \ldots < \beta_\alpha$.
Interrogate $S_{\alpha+1}$ with $\bar{a}_{i,j}$ and obtain $r_{\alpha+1}$.
Use $Knapsack_\alpha((\beta_1, r_{\alpha+1}), \beta_2, \ldots, \beta_\alpha)$ to recover polynomial R.
return R

7 Conclusions

We extended the framework originally proposed in [34] beyond the bivariate setting, demonstrating that the previous recovery techniques can be effectively adapted to the setting of multivariate polynomials constructed through homomorphic operations. Two different strategies were introduced: one leveraging Lagrange's interpolation method, and another grounded in lattice methods tailored to modular knapsack-type problems. As shown in Table 3, a notable contribution of this work is the reduction, under certain conditions, in the number of queries required from the victim from the binomial quantity $\rho(m) + 1$, required by classical multivariate interpolation, to $\deg_{a_1}(P)+2$ queries (Algorithm 3) or 2 queries (Algorithm 5). Together, these contributions provide a broader and more versatile toolkit, while also offering deeper insight into the structural properties that govern these polynomial recovery processes.

Future Work. As a direction for future research, adapting Newton's method to operate in a manner similar to Lagrange interpolation may lead to a more efficient recovery algorithm[8]. Rather than recomputing the entire interpolant, Newton's method updates only the terms associated with newly added or modified points, thereby preserving the previously computed structure. This contrasts with Lagrange interpolation, which, despite its conceptual simplicity, requires a complete reconstruction of the polynomial whenever the set of interpolation points changes.

Another promising direction is analyzing the applicability of the PSLQ algorithm [18]. In particular, we believe it is interesting to see whether PSLQ can outperform certain approaches based on lattice reduction in some specific contexts.

An additional compelling avenue for investigation is represented by artificial intelligence-based methods for solving the problem depicted in Section 2.2.

References

1. Sequence A007018. https://oeis.org/A007018
2. fpLLL. https://github.com/fplll/fplll (2022)
3. Aho, A., Sloane, N.: Some doubly exponential sequences. Fibonacci Quarterly **11(4)**, 429–437 (1973)
4. Balny, S., Delaplace, C., Dequen, G.: Another l makes it better? lagrange meets lll and may improve bkz pre-processing. In: 2025 Proceedings of the Symposium on Algorithm Engineering and Experiments (ALENEX). pp. 181–193. SIAM (2025)
5. Becker, A., Coron, J.S., Joux, A.: Improved Generic Algorithms for Hard Knapsacks. In: EUROCRYPT'11. pp. 364–385. Lecture Notes in Computer Science, Springer (2011)
6. Bos, J.W., Lauter, K.E., Loftus, J., Naehrig, M.: Improved Security for a Ring-Based Fully Homomorphic Encryption Scheme. In: IMACC'13. Lecture Notes in Computer Science, vol. 8308, pp. 45–64. Springer (2013)
7. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: Fully Homomorphic Encryption without Bootstrapping. Cryptology ePrint Archive, Paper 2011/277 (2011), https://eprint.iacr.org/2011/277.pdf
8. Brakerski, Z.: Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP. In: CRYPTO'12. Lecture Notes in Computer Science, vol. 7417, pp. 868–886. Springer (2012)
9. Brakerski, Z., Vaikuntanathan, V.: Lattice-Based FHE as Secure as PKE. In: ITCS'14. p. 1–12. Association for Computing Machinery (2014)
10. Carpov, S., Chillotti, I., Gama, N., Georgieva, M., Izabachene, M.: TFHE: Fast Fully Homomorphic Encryption Library (2016), https://tfhe.github.io/tfhe
11. Cheon, J.H., Kim, A., Kim, M., Song, Y.S.: Homomorphic Encryption for Arithmetic of Approximate Numbers. In: ASIACRYPT'17. Lecture Notes in Computer Science, vol. 10624, pp. 409–437. Springer (2017)
12. Cianciullo, L., Ghodosi, H.: Unconditionally Secure Distributed Oblivious Polynomial Evaluation. In: ICISC 2018. Lecture Notes in Computer Science, vol. 11396, pp. 132–142. Springer (2018)

[8] Note that a multivariate extension of Newton's method can be found in [36].

13. Coster, M.J., Joux, A., LaMacchia, B.A., Odlyzko, A.M., Schnorr, C.P., Stern, J.: Improved Low-Density Subset Sum Algorithms. Comput. Complex. **2**, 111–128 (1992)
14. David, B., Dowsley, R., Katti, R., Nascimento, A.C.: Efficient Unconditionally Secure Comparison and Privacy Preserving Machine Learning Classification Protocolss. In: ProvSec 2015. Lecture Notes in Computer Science, vol. 9451, pp. 354–367. Springer (2015)
15. Dertouzos, M.L., Rivest, R.L., Adleman, L.: On data banks and privacy homomorphisms. Foundations of Secure Computation, Academia Press pp. 169–179 (1978)
16. van Dijk, M., Gentry, C., Halevi, S., Vaikuntanathan, V.: "fully homomorphic encryption over the integers". In: EUROCRYPT'10. Lecture Notes in Computer Science, vol. 6110, pp. 24–43. Springer (2010)
17. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. IACR Cryptol. ePrint Arch. p. 144 (2012), http://eprint.iacr.org/2012/144
18. Ferguson, H.R.P., Bailey, D.H., Kutler, P.: A Polynomial Time, Numerically Stable Integer Relation Algorithm (1992)
19. Freedman, M.J., Ishai, Y., Pinkas, B., Reingold, O.: Keyword Search and Oblivious Pseudorandom Functions. In: TCC 2005. Lecture Notes in Computer Science, vol. 3378, pp. 303–324. Springer (2005)
20. Garey, M.R., Johnson, D.S.: Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co. (1990)
21. Geelen, R., Vercauteren, F.: Fully Homomorphic Encryption for Cyclotomic Prime Moduli. In: EUROCRYPT 2025. pp. 366–397. Lecture Notes in Computer Science, Springer (2025)
22. Gentry, C.: A Fully Homomorphic Encryption Scheme. Ph.D. thesis, Stanford University (2009), https://crypto.stanford.edu/craig/
23. Gentry, C., Sahai, A., Waters, B.: "homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based". In: CRYPTO'13. Lecture Notes in Computer Science, vol. 8042, pp. 75–92. Springer (2013)
24. Gilboa, N.: Two Party RSA Key Generation. In: CRYPTO 1999. Lecture Notes in Computer Science, vol. 1666, pp. 116–129. Springer (1999)
25. Guo, L., Fang, Y., Li, M., Li, P.: Verifiable Privacy-Preserving Monitoring for Cloud-Assisted mHealth Systems. In: INFOCOM 2015. pp. 1026–1034. IEEE (2015)
26. Hamidi, A., Ghodosi, H.: Verifiable DOPE from Somewhat Homomorphic Encryption, and the Extension to DOT. In: SciSec 2022. Lecture Notes in Computer Science, vol. 13580, pp. 105–120. Springer (2022)
27. Hamidi, A., Ghodosi, H.: Fair Distributed Oblivious Polynomial Evaluation via Bitcoin Deposits: Compute-as-a-Service. In: NordSec 2023. Lecture Notes in Computer Science, vol. 14324, pp. 73–86. Springer (2023)
28. Hoffstein, J., Pipher, J., Silverman, J.: An Introduction to Mathematical Cryptography. Undergraduate Texts in Mathematics, Springer (2008)
29. Howgrave-Graham, N., Joux, A.: New Generic Algorithms for Hard Knapsacks. In: Gilbert, H. (ed.) EUROCRYPT'10. pp. 235–256. Springer (2010)
30. Kincaid, D., Kincaid, D.R., Cheney, E.W.: Numerical Analysis: Mathematics of Scientific Computing, vol. 2. American Mathematical Soc. (2009)
31. Lagarias, J.C., Odlyzko, A.M.: Solving Low-Density Subset Sum Problems. J. ACM **32**(1), 229–246 (jan 1985)

32. Li, B., Micciancio, D.: "on the security of homomorphic encryption on approximate numbers". In: EUROCRYPT'21. Lecture Notes in Computer Science, vol. 12696, pp. 648–677. Springer (2021)
33. Li, H.D., Yang, X., Feng, D.G., Li, B.: Distributed Oblivious Function Evaluation and Its Applications. Journal of Computer Science and Technology **19**(6), 942–947 (2004)
34. Maimuţ, D., Teşeleanu, G.: Inferring Bivariate Polynomials for Homomorphic Encryption Application. Cryptography **7**(2), 31 (2023)
35. Martello, S., Toth, P.: Knapsack Problems: Algorithms and Computer Implementations. John Wiley & Sons, Inc. (1990)
36. Micchelli, C.A.: Algebraic Aspects of Interpolation. In: Proceedings of Symposia in Applied Mathematics. vol. 36, pp. 81–102. AMS (1986)
37. Naor, M., Pinkas, B.: Oblivious Transfer and Polynomial Evaluation. In: STOC 1999. pp. 245–254. ACM (1999)
38. Naor, M., Pinkas, B.: Oblivious Polynomial Evaluation. SIAM Journal on Computing **35**(5), 1254–1281 (2006)
39. Nguyen, P.Q., Stehlé, D.: An LLL Algorithm with Quadratic Complexity. SIAM Journal on Computing **39**(3), 874–903 (2009)
40. Otsuka, A., Imai, H.: Unconditionally Secure Electronic Voting. Towards Trustworthy Elections: New Directions in Electronic Voting **6000**, 107–123 (2010)
41. Pouly, A., Shen, Y.: Discrete gaussian sampling for bkz-reduced basis. In: International Conference on Post-Quantum Cryptography. pp. 63–88. Springer (2025)
42. Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P.: Numerical Recipes: The Art of Scientific Computing. Cambridge University Press (2007)
43. Saniee, K.: A Simple Expression for Multivariate Lagrange Interpolation. SIAM undergraduate research online **1**(1), 1–9 (2008)
44. Schnorr, C.: A Hierarchy of Polynomial Time Lattice Basis Reduction Algorithms. Theoretical Computer Science **53**(2), 201–224 (1987)
45. Schroeppel, R., Shamir, A.: A $T = O(2^{n/2})$, $S = O(2^{n/4})$ Algorithm for Certain NP-Complete Problems. SIAM J. Comput. **10**(3), 456–464 (aug 1981)

A Algorithms LO and CJLOSS

Algorithm 7: LO shortest vector oracle.

Input: A vector $a = (a_1, \ldots, a_n)$ of positive integers and an integer S.
Output: A feasible solution $e = (e_1, \ldots, e_n, 0)$ to a knapsack in accordance with Definition 1.

1 Take the following vectors as a basis $[b_1, ..., b_{n+2}]$ for an $n+2$-dimensional integer lattice L:

$$\begin{aligned} b_1 &= (1, 0, \ldots, 0, -a_1) \\ b_2 &= (0, 1, \ldots, 0, -a_2) \\ &\ldots \\ b_n &= (0, \ldots, 1, 0, -a_n) \\ b_{n+1} &= (0, \ldots, 0, 1, S) \end{aligned}$$

Find a reduced basis $[b_1^*, ..., b_{n+2}^*]$ of L using the LLL algorithm.
2 Check if any $b_i^* = (b_{i,1}^*, ..., b_{i,n+2}^*)$ has all $b_{i,j}^* = 0$ or λ for some fixed λ for $1 \leq j \leq n$. For any such b_i^* check whether $e_j = \lambda^{-1} b_{i,j}^*$ for $1 \leq j \leq n$ gives a solution to the knapsack, and if so, stop. Otherwise, continue.
3 Repeat steps 1-3 with S replaced by $S' = \sum_{i=1}^{n} a_i - S$, then stop.

The lattice L for which we aim at finding a reduced basis represents the main difference between the LO shortest vector oracle [31] and the optimized version of it, CJLOSS [13].

More precisely, the vector $b_{n+1} = (0, \ldots, 0, S)$ is replaced by $b_{n+1}^{'} = \left(\frac{1}{2}, \ldots, \frac{1}{2}, S\right)$. In the case of LO, the solution vector of the knapsack problem was in L but for sc cjloss is not. The vector $\overrightarrow{e} = (e_1, ..., e_n, 0)$ becomes $\overrightarrow{e'} = (e_1 - \frac{1}{2}, ..., e_n - \frac{1}{2}, 0)$.

Modifying the shortest vector algorithm used in LO and its variant described in [13] for solving MKP, boils down to a natural modification: the modulus M as an input and a vector $b_{n+2} = (0, \ldots, M)$ added in the lattice basis.