Provably Solving the Hidden Subset Sum Problem via Statistical Learning

Jean-Sébastien Coron and Agnese Gini

University of Luxembourg

Abstract. At Crypto '99, Nguyen and Stern described a lattice based algorithm for solving the hidden subset sum problem, a variant of the classical subset sum problem where the n weights are also hidden. As an application, they showed how to break the Boyko *et al.* fast generator of random pairs $(x, g^x \pmod p)$. The Nguyen-Stern algorithm works quite well in practice for moderate values of n, but its complexity is exponential in n. A polynomial-time variant was recently described at Crypto 2020, based on a multivariate technique, but the approach is heuristic only. In this paper, we describe a proven polynomial-time algorithm for solving the hidden subset-sum problem, based on statistical learning. In addition, we show that the statistical approach is also quite efficient in practice: using the FastICA algorithm, we can reach $n = 250$ in reasonable time.

1 Introduction

The hidden subset-sum problem. At Crypto '99, Nguyen and Stern described a lattice-based algorithm for solving the hidden subset sum problem [NS99], with an application to the cryptanalysis of the fast generator of random pairs $(x, g^x \pmod p)$ from Boyko *et al.* from Eurocrypt '98 [BPV98]. The hidden subset sum problem is a variant of the classical subset sum problem where the n weights α_i are also hidden.

Definition 1 (Hidden Subset Sum Problem). *Let q be an integer, and let $\alpha_1, \ldots, \alpha_n$ be random integers in $\mathbb{Z}_q$. Let $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{Z}^m$ be random vectors with components in $\{0, 1\}$. Let $\mathbf{h} = (h_1, \ldots, h_m) \in \mathbb{Z}^m$ satisfying:*

$$\mathbf{h} = \alpha_1 \mathbf{x}_1 + \alpha_2 \mathbf{x}_2 + \cdots + \alpha_n \mathbf{x}_n \pmod q \tag{1}$$

Given q and $\mathbf{h}$, recover the vector $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$ and the vectors $\mathbf{x}_i$'s, up to a permutation of the α_i's and $\mathbf{x}_i$'s.

Recall that the classical subset sum problem with known weights α_i's can be solved in polynomial time by a lattice based algorithm [LO85], when the density $d = n/\log q$ is $\mathcal{O}(1/n)$. Provided a shortest vector oracle, the classical subset sum problem can be solved when the density d is less than $\simeq 0.94$. The algorithm is based on finding a shortest vector in a lattice built from $h, \alpha_1, \ldots, \alpha_n, q$; see [CJL+92]. For the hidden subset sum problem, the attack is clearly not applicable since the weights α_i's are hidden.

The Nguyen-Stern algorithm. For solving the hidden subset-sum problem, the Nguyen-Stern algorithm relies on the technique of the orthogonal lattice. If a vector $\mathbf{u}$ is orthogonal modulo q to the public vector of samples $\mathbf{h}$, then from (1) we must have:

$$\langle \mathbf{u}, \mathbf{h} \rangle \equiv \alpha_1 \langle \mathbf{u}, \mathbf{x}_1 \rangle + \cdots + \alpha_n \langle \mathbf{u}, \mathbf{x}_n \rangle \equiv 0 \pmod q$$

This implies that the vector $\mathbf{p_u} = (\langle \mathbf{u}, \mathbf{x}_1 \rangle, \ldots, \langle \mathbf{u}, \mathbf{x}_n \rangle)$ is orthogonal to the hidden vector $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$ modulo q. Now, if the vector $\mathbf{u}$ is short enough, the vector $\mathbf{p_u}$ will be short (since the vectors $\mathbf{x}_i$ have components in $\{0, 1\}$ only), and if $\mathbf{p_u}$ is shorter than the shortest vector orthogonal to $\boldsymbol{\alpha}$ modulo q, we must have $\mathbf{p_u} = 0$, and therefore the vector $\mathbf{u}$ will be orthogonal in $\mathbb{Z}$ to all vectors $\mathbf{x}_i$. The orthogonal lattice attack consists in generating with LLL many short vectors $\mathbf{u}$

orthogonal to $\mathbf{h}$; this reveals the lattice of vectors orthogonal to the $\mathbf{x}_i$'s, and eventually the lattice $\mathcal{L}_{\mathbf{x}}$ generated by the vectors $\mathbf{x}_i$'s. In a second step, by finding sufficiently short vectors in the lattice $\mathcal{L}_{\mathbf{x}}$, one can recover the original vectors $\mathbf{x}_i$'s, and eventually the hidden weight $\boldsymbol{\alpha}$ by solving a linear system.

While the Nguyen-Stern algorithm works quite well in practice for moderate values of n, its complexity is actually exponential in the number of weights n; see [CG20]. Namely, in the first step one only recovers a basis of the lattice $\mathcal{L}_{\mathbf{x}}$ generated by the binary vectors $\mathbf{x}_i$, but not necessarily the original vectors $\mathbf{x}_i$'s, because the basis vectors that we recover via LLL can be much larger than the $\mathbf{x}_i$'s. In order to recover the $\mathbf{x}_i$'s, in a second step one must therefore compute a very short basis of the n-dimensional lattice $\mathcal{L}_{\mathbf{x}}$. In principle, this takes exponential-time in n, as one must apply BKZ reduction [Sch87] with increasingly large block-sizes; this was confirmed by practical experiments in [CG20].

Cryptographic application. As an application, the authors of [NS99] showed how to break the fast generator of random pairs $(x, g^x \pmod p)$ from Boyko, Peinado and Venkatesan from Eurocrypt '98 [BPV98]. Such generator can be used to speed-up the generation of discrete-log pairs with fixed base g, as in Schnorr identification, and in Schnorr, ElGamal and DSS signatures. Namely, for a prime number p and for $g \in \mathbb{Z}_p^*$ of order q, the generator first precomputes n values $\beta_j = g^{\alpha_j}$ for random $\alpha_j \in \mathbb{Z}_q$; it can then repeatedly generate a random $x = \sum_{i=1}^{n} \alpha_i \cdot x_i \bmod q$ for $x_i \leftarrow \{0, 1\}$, with the corresponding group element:

$$g^x = \prod_{i=1}^{n} \beta_j^{x_i} \bmod p. \tag{2}$$

This requires on average $n/2$ multiplications where n is the number of hidden weights α_j, instead of a full exponentiation in $\mathbb{Z}_p^*$.

The Coron-Gini algorithm. At Crypto 2020, Coron and Gini described a variant of the Nguyen-Stern algorithm for solving the hidden subset sum problem that works in heuristic polynomial-time. The first step is still the same orthogonal lattice attack with LLL as in Nguyen-Stern. In the second step, instead of applying BKZ, the authors use a multivariate technique that recovers the short lattice vectors and finally the hidden secrets in polynomial time. However, such multivariate approach requires $m \simeq n^2/2$ samples instead of $m = 2n$ as in [NS99]. Asymptotically, the heuristic complexity of the full algorithm is $\mathcal{O}(n^9)$. The authors performed some practical experiments to compare it with the BKZ approach; with the new multivariate approach they could reach $n = 250$ in a reasonable amount of time, instead of $n = 170$ with BKZ. The multivariate algorithm from [CG20] enables to break the Boyko *et al.* generator in polynomial-time instead of exponential-time as in the original Nguyen-Stern attack, albeit with a significantly larger number of samples from the generator, namely $m \simeq n^2/2$ samples instead of $m = 2n$, respectively.

Our contribution. Our main contribution is to describe a proven polynomial-time algorithm for solving the hidden subset-sum problem, while the Coron-Gini algorithm was heuristic only. Our algorithm proceeds in two steps. The first step is the same as in Nguyen-Stern's algorithm, and reveals a basis of the completed lattice $\bar{\mathcal{L}}_{\mathbf{x}}$ generated by the n vectors $\mathbf{x}_i$, via an orthogonal lattice attack. In the second step, we use a statistical learning technique to disclose the hidden vectors and weights, using an approach introduced by Nguyen and Regev for the cryptanalysis of GGH and NTRU signatures [NR09]. Indeed, our main observation is that from a basis of $\bar{\mathcal{L}}_{\mathbf{x}}$ it is possible to derive a set of samples from an unknown discrete distribution, and identifying such distribution solves the hidden subset-sum problem.

We also describe a heuristic extension to a variant of the hidden subset-sum problem where the coefficients $\mathbf{x}_i$'s lie in a discrete interval $[0, B] \cap \mathbb{Z}$ instead of $\{0, 1\}$; we call this the *hidden*

linear combination problem. This corresponds to a generalisation of the Boyko *et al.* pseudo-random generator of discrete-log pairs $(g, g^x \pmod p)$, where in Equation (2) the exponents x_i lie in $[0, B] \cap \mathbb{Z}$ instead of $\{0, 1\}$; this allows to increase the entropy of the generator, for a fixed number n of pre-computed group elements β_j. The original Nguyen-Stern algorithm can still be adapted to the generalised problem, and its heuristic complexity remains exponential in n, as in the binary case, and polynomial in $\log B$. For this variant the multivariate approach from [CG20] does not apply, because it would lead to multivariate polynomials of degree $B + 1$; namely, the technique would remain polynomial-time only for constant B, and be essentially unpractical. We argue that for the generalised problem our statistical approach has heuristic complexity polynomial in n and B. We summarise in Table 1 the algorithm complexities.

		complexity	status
Hidden subset sum ($B = 1$)	Nguyen-Stern [NS99]	$2^{\Omega(n)}$	heuristic
	Coron-Gini [CG20]	$\mathcal{O}(n^9)$	heuristic
	Statistical attack	$\mathsf{poly}(n)$	proven
Hidden linear combination	Nguyen-Stern [NS99]	$2^{\Omega(n)} \cdot \log^{\mathcal{O}(1)} B$	heuristic
	Statistical attack	$\mathsf{poly}(n, B)$	heuristic

Table 1. Algorithmic complexity for solving the hidden subset sum problem ($B = 1$) and the hidden linear combination problem.

Practical attack. We show that the statistical approach is also quite efficient in practice when based on FastICA [HO97], which is an algorithm to solve the signal source separation problem in the context of Independent Component Analysis (ICA). Namely, we describe a practical implementation using $m \simeq n^2$ samples as in [CG20], but with space complexity $\mathcal{O}(n^3)$ instead of $\mathcal{O}(n^4)$. We provide the source code in:

https://pastebin.com/WzGXHmpW

2 Background on lattices

Lattices and bases. Let $\mathbf{b}_1, \ldots, \mathbf{b}_k \in \mathbb{Z}^m$ be linearly independent vectors for $k \leq m$. The (integral) *lattice* generated by the basis $\mathfrak{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_k\}$ is

$$\mathcal{L}(\mathfrak{B}) = \left\{ \sum_{i=1}^{k} v_i \mathbf{b}_i \;\mid\; v_1, \ldots, v_k \in \mathbb{Z} \right\}.$$

A matrix $\mathbf{B}$ whose rows are a basis of a lattice is called *base matrix*. It is possible to prove that two basis $\mathbf{B}, \mathbf{B}'$ generate the same lattice if and only if there exists a unimodular matrix $\mathbf{U} \in \mathrm{GL}_k(\mathbb{Z})$ such that $\mathbf{UB} = \mathbf{B}'$. Given any basis $\mathbf{B}$ its Gram-determinant is $d(\mathbf{B}) = \sqrt{\det(\mathbf{B}\mathbf{B}^\intercal)}$; then this number is invariant under base change. Thus, the *determinant* of a lattice $\mathcal{L}$, i.e. the Gram-determinant of any of its basis $\mathbf{B}$, $\det(\mathcal{L}) = d(\mathbf{B})$ is well defined. The *dimension* $\dim(\mathcal{L})$, or *rank*, of a lattice is the dimension as vector space of $E_{\mathcal{L}} \coloneqq \mathrm{Span}_{\mathbb{R}}(\mathcal{L})$. If $\mathcal{L}' \subseteq \mathcal{L}$, then we say that $\mathcal{L}'$ is a *sublattice* of a lattice $\mathcal{L}$ and $\mathcal{L}$ is a *superlattice* of $\mathcal{L}'$. $\mathcal{L}'$ is a *full-rank* sublattice of $\mathcal{L}$ if $\dim(\mathcal{L}') = \dim(\mathcal{L})$. In particular, if $\mathcal{L} \subseteq \mathbb{Z}^m$ is a full-rank sublattice, we simply say that $\mathcal{L}$ is full-rank. For a full-rank sublattice it holds $\det(\mathcal{L}) \leq \det(\mathcal{L}')$.

Orthogonal lattice. The *orthogonal lattice* of a lattice $\mathcal{L} \subseteq \mathbb{Z}^m$ is

$$\mathcal{L}^\perp \coloneqq \{\mathbf{v} \in \mathbb{Z}^m \mid \forall \mathbf{b} \in \mathcal{L},\ \langle \mathbf{v}, \mathbf{b} \rangle = 0\} = E_{\mathcal{L}}^\perp \cap \mathbb{Z}^m$$

where $\langle \cdot, \cdot \rangle$ denotes the standard scalar product of $\mathbb{R}^m$. The *completion* of a lattice $\mathcal{L}$ is the lattice $\bar{\mathcal{L}} = E_{\mathcal{L}} \cap \mathbb{Z}^m = (\mathcal{L}^\perp)^\perp$. Moreover, a lattice is called *complete* if it coincides with its completion, i.e. $\bar{\mathcal{L}} = \mathcal{L}$. Notice that $\mathcal{L}$ is a full-rank sublattice of $\bar{\mathcal{L}}$, and recall $\dim \mathcal{L} + \dim \mathcal{L}^\perp = m$ and $\det(\mathcal{L}^\perp) = \det(\bar{\mathcal{L}}) \leq \det(\mathcal{L})$ (proofs of these facts are recalled in [CG20, Appendix A]). By Hadamard's inequality, we have $\det(\mathcal{L}) \leq \prod_{\mathbf{b} \in b} \|\mathbf{b}\|$ for any basis B of $\mathcal{L}$; this implies that $\det(\mathcal{L}^\perp) \leq \prod_{\mathbf{b} \in B} \|\mathbf{b}\|$.

Lattice minima. For each $1 \leq i \leq \dim \mathcal{L}$, the *i-th minimum* $\lambda_i(\mathcal{L})$ of a lattice $\mathcal{L}$ is the minimum of the $\max_j \{\|\mathbf{v}_j\|\}$ among all sets $\{\mathbf{v}_j\}_{j \leq i}$ of i linearly independent lattice points. Notice that the first minimum $\lambda_1(\mathcal{L})$ is the minimum of the norm of its non-zero vectors; hence, the lattice points whose norm is $\lambda_1(\mathcal{L})$ are called *shortest vectors*, accordingly.

The *Hermite constant* γ_k (in dimension k) is the supremum of $\lambda_1(\mathcal{L})^2 / \det(\mathcal{L})^{\frac{2}{k}}$ over all the lattices of rank k. Using Minkowski convex body theorem, one can prove that for each $k \in \mathbb{N}^+$, $0 \leq \gamma_k \leq k/4 + 1$. This constant is also involved in the *Minkowski's Second Theorem*, which asserts that for each $1 \leq i \leq k$

$$\left(\prod_{j=1}^{i} \lambda_i(\mathcal{L}) \right)^{\frac{1}{i}} \leq \sqrt{\gamma_k} \det(\mathcal{L})^{\frac{1}{k}}.$$

Lattice reduction. The notion of LLL-reduced basis was introduced in [LLL82], along with an algorithm to produce such bases. Those have many good proprieties, for instance the first vector of an LLL-reduced basis is not much longer than the shortest vector of the lattice.

Lemma 1 (LLL-reduced basis). *Let $\mathbf{b}_1, \ldots, \mathbf{b}_k \in \mathbb{Z}^m$ an LLL-reduced basis of a lattice $\mathcal{L}$. Then $\|\mathbf{b}_1\| \leq 2^{\frac{k-1}{2}} \lambda_1(\mathcal{L})$, and $\|\mathbf{b}_j\| \leq 2^{\frac{k-1}{2}} \lambda_i(\mathcal{L})$ for each $1 \leq j \leq i \leq k$.*

The LLL algorithm produces an LLL-reduced basis in time $\mathcal{O}(k^5 m \log^3 \beta)$, given a basis of vectors of norm less than β of a k-rank sublattice of $\mathbb{Z}^m$. Nguyen and Stehlé in [NS09] proposed a variant, based on proven floating point arithmetic and called L^2, whose complexity is $\mathcal{O}(k^4 m(k + \log \beta) \log \beta)$ without fast arithmetic. In this paper, when we apply LLL, we always mean the L^2 variant. There exist other notions of reduced basis. If a better approximation of the shortest vector is needed, one can use Schnorr's algorithm BKZ [Sch87]. In this paper, when we apply BKZ, we generally refer to BKZ 2.0 [CN11].

Heuristics. By the *Gaussian Heuristic*, for a " random lattice" we can expect $\lambda_1(\mathcal{L}) \approx \sqrt{k} \det(\mathcal{L})^{\frac{1}{k}}$. In such case we can also expect that all lattice minima have approximately the same value. In general, we can suppose that a lattice $\mathcal{L}$ generated by a set of k "random" vectors in $\mathbb{Z}^m$ for $k < m$ has rank k, and that the short vectors of $\mathcal{L}^\perp$ have norm approximately $(\det \mathcal{L}^\perp)^{1/(m-k)} \simeq (\det \mathcal{L})^{1/(m-k)} \simeq (\prod_{i=1}^{k} \|\mathbf{b}_i\|)^{1/(m-k)}$, up to a $\sqrt{k}$ factor.

3 The Nguyen-Stern algorithm

We recall the Nguyen-Stern algorithm [NS99] for solving the hidden subset-sum. We also recall the complexity analysis from [CG20]. In the hidden subset-sum problem, we are given a modulus q and $\mathbf{h} = (h_1, \ldots, h_m) \in \mathbb{Z}^m$ satisfying

$$\mathbf{h} = \alpha_1 \mathbf{x}_1 + \alpha_2 \mathbf{x}_2 + \cdots + \alpha_n \mathbf{x}_n \pmod{q} \tag{3}$$

and we must recover the vector $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n) \in \mathbb{Z}_q^n$ and the vectors $\mathbf{x}_i \in \{0, 1\}^m$. The Nguyen-Stern algorithm comprises two steps:

1. From the samples $\mathbf{h}$, determine the lattice $\bar{\mathcal{L}}_{\mathbf{x}}$, where $\mathcal{L}_{\mathbf{x}}$ is the lattice generated by the $\mathbf{x}_i$'s.
2. From $\bar{\mathcal{L}}_{\mathbf{x}}$, recover the hidden vectors $\mathbf{x}_i$'s using BKZ. From $\mathbf{h}$, the $\mathbf{x}_i$'s and q, recover the weights α_i.

3.1 First step: orthogonal lattice attack

In the first step the goal is to recover $\bar{\mathcal{L}}_{\mathbf{x}}$ from $\mathbf{h}$ and q. Let $\mathcal{L}_0$ be the lattice of vectors orthogonal to $\mathbf{h}$ modulo q:

$$\mathcal{L}_0 \coloneqq \Lambda_q^{\perp}(\mathbf{h}) = \{\mathbf{u} \in \mathbb{Z}^m \mid \langle \mathbf{u}, \mathbf{h} \rangle \equiv 0 \pmod{q} \}$$

For any $\mathbf{u} \in \mathcal{L}_0$, the vector

$$\mathbf{p}_{\mathbf{u}} = (\langle \mathbf{u}, \mathbf{x}_1 \rangle, \ldots, \langle \mathbf{u}, \mathbf{x}_n \rangle)$$

is orthogonal to the vector $\boldsymbol{\alpha}$ modulo q, since from (3) we obtain:

$$\langle \mathbf{u}, \mathbf{h} \rangle \equiv \alpha_1 \langle \mathbf{u}, \mathbf{x}_1 \rangle + \cdots + \alpha_n \langle \mathbf{u}, \mathbf{x}_n \rangle \equiv 0 \pmod{q}.$$

Then, if $\mathbf{p}_{\mathbf{u}}$ is shorter than the shortest non-zero vector orthogonal to $\boldsymbol{\alpha}$ modulo q, we must have $\mathbf{p}_{\mathbf{u}} = 0$, and therefore $\mathbf{u} \in \mathcal{L}_{\mathbf{x}}^{\perp}$. Therefore, the orthogonal lattice attack first obtains an LLL-reduced basis of $\mathcal{L}_0$, from which it extracts a generating set for $\mathcal{L}_{\mathbf{x}}^{\perp}$; subsequently, it computes the orthogonal of $\mathcal{L}_{\mathbf{x}}^{\perp}$ obtaining $\bar{\mathcal{L}}_{\mathbf{x}}$.

Algorithm 1 Orthogonal lattice attack

Input: $\mathbf{h}, q, n$.
Output: A basis of $\bar{\mathcal{L}}_{\mathbf{x}}$.
1: Compute an LLL-reduced basis $\mathbf{u}_1, \ldots, \mathbf{u}_m$ of $\mathcal{L}_0$.
2: Extract a generating set of $\mathbf{u}_1, \ldots, \mathbf{u}_{m-n}$ of $\mathcal{L}_{\mathbf{x}}^{\perp}$.
3: Compute a basis $(\mathbf{c}_1, \ldots, \mathbf{c}_n)$ of $\bar{\mathcal{L}}_{\mathbf{x}} = (\mathcal{L}_{\mathbf{x}}^{\perp})^{\perp}$.
4: **return** $(\mathbf{c}_1, \ldots, \mathbf{c}_n)$

The orthogonal lattice attack described in Algorithm 1 is guaranteed to succeed with good probability for sufficiently large q.

Theorem 1. *([CG20, Theorem 1]) Let $m > n$. Assume that the lattice $\mathcal{L}_{\mathbf{x}}$ has rank n. With probability at least $1/2$ over the choice of $\boldsymbol{\alpha}$, Algorithm 1 recovers a basis of $\bar{\mathcal{L}}_{\mathbf{x}}$ in polynomial time, assuming that q is a prime integer of bitsize at least $2mn \log m$. For $m = 2n$, the density of the subset-sum problem is $d = n/\log q = \mathcal{O}(1/(n \log n))$.*

Heuristic analysis. One can use a slightly smaller value of the modulus q via a heuristic analysis. As shown in [CG20], for $m = 2n$ samples, we can use $\log q = \mathcal{O}(n^2)$, which gives a knapsack density $d = n/\log q = \mathcal{O}(1/n)$ as in the classical subset-sum problem. More concretely, one can take $\log q \simeq 2\iota \cdot n^2 + n \log n$ with $\iota = 0.035$.

Orthogonal lattice attack for large m. The multivariate attack described in [CG20] requires a much larger number of samples, namely $m \simeq n^2$ instead of $m = 2n$ in Nguyen-Stern. In our statistical attack (Section 4), we also require a similarly large number of samples. For such large value of m, it would not be efficient to apply Algorithm 1 directly, as the lattice dimension of the lattice $\mathcal{L}_0$ would be too large, namely $m \simeq n^2$. Instead, a better approach is to apply LLL only to the first $2n$ coordinates, which already gives n orthogonal vectors, and then compute the other $m - 2 \cdot n$ orthogonal vectors using size reduction. We refer to [CG20] for the details.

3.2 Second step: the BKZ approach

The first step of the Nguyen-Stern algorithm produces an LLL-reduced basis $(\mathbf{c}_1, \ldots, \mathbf{c}_n)$ of the completed lattice $\bar{\mathcal{L}}_{\mathbf{x}} \subset \mathbb{Z}^m$. Due to the LLL approximation factor, the recovered basis vectors $(\mathbf{c}_1, \ldots, \mathbf{c}_n)$ can be much larger than the original vectors $\mathbf{x}_i$, which are among the shortest vectors in $\mathcal{L}_{\mathbf{x}}$. Therefore, in the second step BKZ is applied to recover the original vectors $\mathbf{x}_i$. Indeed, BKZ provides a better approximation factor than LLL. More precisely, the analysis in [CG20] shows that the BKZ approximation factor $2^{\iota \cdot n}$ must satisfy $2^{\iota \cdot n} \leq \sqrt{n}/2$. Since achieving an Hermite factor of $2^{\iota \cdot n}$ heuristically requires time $2^{\Omega(1/\iota)}$ with block-size $\beta = \omega(1/\iota)$, the resulting heuristic running time of the Nguyen-Stern algorithm is $2^{\Omega(n/\log n)}$, with BKZ block-size $\beta = \omega(n/\log n)$. Eventually, from the vector $\mathbf{h}$, the $\mathbf{x}_i$'s and q, one can recover the hidden weights α_i by solving a linear system.

4 Our statistical algorithm for hidden subset-sums

In this section we describe our algorithm for solving the hidden subset-sum problem based on a statistical learning technique.

As recalled in Section 3, the Nguyen-Stern algorithm comprises two steps.

1. From the samples $\mathbf{h}$, determine the lattice $\bar{\mathcal{L}}_{\mathbf{x}}$, where $\mathcal{L}_{\mathbf{x}}$ is the lattice generated by the $\mathbf{x}_i$'s.
2. From $\bar{\mathcal{L}}_{\mathbf{x}}$, recover the hidden vectors $\mathbf{x}_i$'s using BKZ. From $\mathbf{h}$, the $\mathbf{x}_i$'s and q, recover the weights α_i.

Our algorithm uses the same first step as in Nguyen-Stern, but we replace BKZ in the second step by a statistical approach. Let $\mathbf{X} \in \mathbb{Z}^{n \times m}$ be the binary matrix of row vectors $\mathbf{x}_i \in \mathbb{Z}^m$, whose components are randomly distributed in $\{0, 1\}$. From the first step, by Theorem 1, we obtain a basis of the lattice $\bar{\mathcal{L}}_{\mathbf{x}}$, namely a matrix of row vectors $\mathbf{C} \in \mathbb{Z}^{n \times m}$ such that

$$\mathbf{C} = \mathbf{V} \cdot \mathbf{X}$$

for some unknown matrix $\mathbf{V} \in \mathrm{GL}_n(\mathbb{Q})$.

For the second step, our main observation is that the m columns of $\mathbf{C}$ are samples from the discrete parallelepiped:

$$\mathcal{P}_{\{0,1\}}(\mathbf{V}) = \left\{ \sum_{i=1}^{n} x_i \mathbf{v}_i \ : \ x_i \in \{0, 1\} \right\}$$

where $\mathbf{v}_1, \ldots, \mathbf{v}_n \in \mathbb{Q}^n$ are the columns of $\mathbf{V}$. Therefore, our approach consists in recovering the matrix $\mathbf{V}$ from those samples, by using a statistical technique. Indeed, given $\mathbf{V}$ one can compute the $\mathbf{x}_i$'s with $\mathbf{X} = \mathbf{V}^{-1}\mathbf{C}$. In the following, we show that the statistical approach allows to solve the hidden subset-sum problem in polynomial time in a provable way.

Continuous and discrete parallelepipeds. The problem of learning a *continuous* parallelepiped was solved by Nguyen and Regev in [NR09] for the cryptanalysis of GGH and NTRU signatures. Given a matrix $\mathbf{V} \in \mathrm{GL}_n(\mathbb{R})$, the continuous parallelepiped is defined as:

$$\mathcal{P}_{[-1,1]}(\mathbf{V}) = \left\{ \sum_{i=1}^{n} x_i \mathbf{v}_i \ : \ x_i \in [-1, 1] \right\}$$

Problem 1 (Hidden Parallelepiped Problem) *Let $\mathbf{V} \in \mathrm{GL}_n(\mathbb{R})$ be a matrix of column vectors $[\mathbf{v}_1, \ldots, \mathbf{v}_n]$ and let $\mathcal{P}_{[-1,1]}(\mathbf{V}) = \{\sum_{i=1}^{n} x_i \mathbf{v}_i \ : \ x_i \in [-1, 1]\}$. The input to the HPP is a sequence of $\mathsf{poly}(n)$ independent samples from $\mathcal{U}(\mathcal{P}_{[-1,1]}(\mathbf{V}))$, the uniform distribution over $\mathcal{P}_{[-1,1]}(\mathbf{V})$. The goal is to recover a good approximation of the columns of $\pm\mathbf{V}$.*

More precisely, the authors proved the following theorem.

Theorem 2 (Nguyen-Regev). *There exists an algorithm $\mathcal{A}$ such that for any $c_0 > 0$, there exists $c_1 > 0$ such that given n^{c_1} samples uniformly distributed over some parallelepiped $\mathcal{P}_{[-1,1]}(\mathbf{V})$, for $\mathbf{V} \in \mathrm{GL}_n(\mathbb{R})$, the algorithm $\mathcal{A}$ returns with constant probability a vector $\mathbf{Ve}$, where $\mathbf{e}$ is within ℓ_2 distance n^{-c_0} of some standard basis vector $\mathbf{e}_i$.*

As stated above, the Nguyen-Regev algorithm recovers an approximation of a single column of $\pm\mathbf{V}$, but it can be easily modified to provably recover an approximation of all columns of $\pm\mathbf{V}$. However, there are two main differences with our case:

- We obtain random samples from the finite parallelepiped set $\mathcal{P}_{\{0,1\}}(\mathbf{V})$, whereas the Nguyen-Regev algorithm uses samples from the continuous set $\mathcal{P}_{[-1,1]}(\mathbf{V})$.
- The Nguyen-Regev algorithm only recovers an approximation of the columns of $\pm\mathbf{V}$ with relative error $1/n^{c_0}$, whereas we must recover the exact value of the columns of $\pm\mathbf{V}$.

Note that in [NR09] for the cryptanalysis of NTRU and GGH signatures, the matrix $\mathbf{V}$ is an integer matrix whose entries are polynomially bounded in absolute value; therefore even with relative error n^{-c_0}, for large enough c_0, the absolute error becomes less than $1/2$ in each coordinate, and the columns of $\pm\mathbf{V}$ can be exactly recovered simply by rounding to the nearest integer. Therefore the authors of [NR09] obtain a rigorous proof that the secret key in NTRU and GGH signatures can be recovered in polynomial time. However, in our case the entries of $\mathbf{V}$ are not polynomially bounded in absolute value.

Nevertheless, in our discrete variant, even if the matrix $\mathbf{V}$ has large entries, we can use the above equation $\mathbf{C} = \mathbf{V} \cdot \mathbf{X}$, and from an approximation of $\mathbf{V}$ obtain an approximation of $\mathbf{X}$; since the matrix $\mathbf{X}$ is binary, we can then recover $\mathbf{X}$ by rounding to the nearest integer, which enables us to eventually recover $\mathbf{V}$. Formally, we consider the following problem:

Problem 2 (Discrete Hidden Parallelepiped Problem) *Let $\mathbf{V} \in \mathrm{GL}_n(\mathbb{Q})$, consider $\mathcal{P}_{\{0,1\}}(\mathbf{V}) \coloneqq \{\mathbf{Vx} |\ \mathbf{x} \in \{0,1\}^n\}$ the discrete parallelepiped associated to $\mathbf{V}$. Given $\mathsf{poly}(n)$ independent samples from the uniform distribution over $\mathcal{P}_{\{0,1\}}(\mathbf{V})$, recover the columns of $\mathbf{V}$.*

Our approach for solving the discrete hidden parallelepiped problem (Problem 2) is to reduce to the continuous hidden parallelepiped problem (Problem 1). Namely, we show that we can easily generate samples from the continuous distribution from samples from the discrete distribution. Then we can use the Nguyen-Regev learning algorithm as a black-box. Clearly, computing the exact value of $\mathbf{V}$ is not really necessary for the hidden subset sum problem, as we are only interested in recovering the vectors $\mathbf{x}_i$'s.

4.1 From discrete to continuous

Our goal is to obtain a set of independent samples of the continuous $\mathcal{P}_{[-1,1]}(\mathbf{V})$ from a set of independent samples of the discrete $\mathcal{P}_{\{0,1\}}(\mathbf{V})$. Consider the continuous distribution $x \leftarrow [0,1]$; we can approximate this distribution up to k-bit precision by generating a random k-bit binary decomposition, *i.e.* we can let $y = \sum_{i=1}^{k-1} b_i 2^{-i}$. To approximate the continuous distribution $x \leftarrow [-1,1]$, we can use one more bit b_0, with $y = -b_0 + \sum_{i=1}^{k-1} b_i 2^{-i}$. Our main observation is that we can proceed similarly with the samples from $\mathcal{P}_{\{0,1\}}(\mathbf{V})$ to generate samples from $\mathcal{P}_{[-1,1]}(\mathbf{V})$. Moreover, to obtain a continuous distribution, we add a small error $\mathbf{e}$.

Formally, given parameters $k \in \mathbb{Z}$ and $\varepsilon > 0$, we define the random vector

$$\mathbf{v} = -\mathbf{u}_0 + \sum_{i=1}^{k} 2^{-i}\mathbf{u}_i + \mathbf{e}$$

where $\mathbf{u}_i \leftarrow \mathcal{U}(\mathcal{P}_{\{0,1\}}(\mathbf{V}))$ for $0 \leq i < k$ and $\mathbf{e} \leftarrow \mathcal{U}([0,\varepsilon)^n)$. We want to show that for large enough k and small enough ε, the distribution of $\mathbf{v}$ is statistically close to uniform in $\mathcal{P}_{[-1,1]}(\mathbf{V})$. Indeed, this implies that we can produce samples from the continuous set $\mathcal{P}_{[-1,1]}(\mathbf{V})$ by combining those from the discrete set $\mathcal{P}_{\{0,1\}}(\mathbf{V})$.

We denote by $\mathcal{D}(\mathbf{V}, k, \varepsilon)$ the distribution of the vector $\mathbf{v}$. By definition, the vector $\mathbf{v}$ is distributed as:

$$\mathbf{v} = \mathbf{V} \cdot \left(-\mathbf{b}_0 + \sum_{i=1}^{k} 2^{-i} \mathbf{b}_i \right) + \mathbf{e}$$

where for all $0 \leq i \leq k$ the components of $\mathbf{b}_i \in \mathbb{Z}^n$ are randomly distributed in $\{0,1\}$, and $\mathbf{e} \leftarrow [0,\varepsilon)^n$. Therefore the vector $\mathbf{v}$ is distributed as $\mathbf{v} = \mathbf{V}\mathbf{y} + \mathbf{e}$, where $\mathbf{e} \leftarrow [0,\varepsilon)^n$ and the components of $\mathbf{y}$ are uniformly distributed in the set:

$$\mathcal{J}_k = \left\{ -b_0 + \sum_{r=1}^{k} b_r 2^{-r} \ : \ (b_0, \ldots, b_k) \in \{0,1\}^{k+1} \right\}.$$

We define the statistical distance between two probability distributions μ and ν on E as

$$\delta(\mu, \nu) = \sup_{A \subset E} |\mu(A) - \nu(A)|$$

Lemma 2. *The statistical distance between $\mathcal{U}(\mathcal{P}_{[-1,1]}(\mathbf{V}))$ and $\mathcal{D}(\mathbf{V}, k, \varepsilon)$ with $\varepsilon(k) = 2^{-k/2}$ is at most $n \cdot 2^{-k/2} \cdot (\|\mathbf{V}\|_\infty + \|\mathbf{V}^{-1}\|_\infty)$.*

Proof. We first consider an intermediate distribution $\mathcal{D}'(\mathbf{V}, \varepsilon)$ of $\mathbf{a} = \mathbf{V}\mathbf{x} + \mathbf{e}$ where $\mathbf{x} \leftarrow [-1,1]^n$ and $\mathbf{e} \in [0,\varepsilon)^n$. We have that $\mathbf{a} = \mathbf{V}(\mathbf{x} + \mathbf{V}^{-1}\mathbf{e})$. For a fixed $\mathbf{u}$ the statistical distance between $\mathbf{x}$ and $\mathbf{x} + \mathbf{u}$ is at most $n \cdot \|\mathbf{u}\|_\infty$. Hence for a fixed $\mathbf{e}$ the statistical distance between $\mathbf{x}$ and $\mathbf{x} + \mathbf{V}^{-1}\mathbf{e}$ is at most $n \cdot \|\mathbf{V}^{-1}\|_\infty \cdot \varepsilon$. Therefore the statistical distance between $\mathcal{U}(\mathcal{P}_{[-1,1]}(\mathbf{V}))$ and $\mathcal{D}'(\mathbf{V}, \varepsilon)$ is also at most $n \cdot \|\mathbf{V}^{-1}\|_\infty \cdot \varepsilon$.

Moreover any $x \in [-1,1]$ can be decomposed as $x = y + d$ where $y \in \mathcal{J}_k$ and $d \in [0, 2^{-k})$. Letting $\mathbf{a}$ be a random vector distributed as $\mathcal{D}'(\mathbf{V}, \varepsilon)$, we can therefore write

$$\mathbf{a} = \mathbf{V}\mathbf{x} + \mathbf{e} = \mathbf{V}\mathbf{y} + \mathbf{V}\mathbf{d} + \mathbf{e}$$

with $\mathbf{y} \leftarrow \mathcal{J}_k^n$, $\mathbf{d} \leftarrow [0, 2^{-k})^n$ and $\mathbf{e} \leftarrow [0,\varepsilon)^n$. As previously, the statistical distance between $\mathbf{V}\mathbf{d} + \mathbf{e}$ and $\mathbf{e}$ is at most $n \cdot \|\mathbf{V}\|_\infty \cdot \varepsilon^{-1} \cdot 2^{-k}$. This implies that the statistical distance between $\mathbf{a} = \mathbf{V}\mathbf{y} + (\mathbf{V}\mathbf{d} + \mathbf{e})$ and $\mathbf{V}\mathbf{y} + \mathbf{e}$ satisfies the same bound. This implies that the statistical distance between $\mathcal{D}'(\mathbf{V}, \varepsilon)$ and $\mathcal{D}(\mathbf{V}, k, \varepsilon)$ is at most $n \cdot \|\mathbf{V}\|_\infty \cdot \varepsilon^{-1} \cdot 2^{-k}$. Finally, combining the two bounds and taking $\varepsilon(k) = 2^{-k/2}$, we obtain the required bound.

4.2 The Nguyen-Regev learning technique

As recalled in Theorem 2, the Nguyen-Regev algorithm recovers an approximation of a column of $\pm\mathbf{V}$, from polynomially many samples from $\mathcal{U}(\mathcal{P}_{[-1,1]}(\mathbf{V}))$. Repeating the algorithm multiple times does not guarantee to recover all the columns. However, this can be fixed by slightly changing the algorithm. Suppose $\mathbf{V}$ is an orthogonal matrix, and that we have computed a set of columns $V = \{\mathbf{v}_1, \ldots, \mathbf{v}_i\}$; we know that the further target vectors must belong to $Span\,(V)^\perp$. Therefore, at each iterative step we can project on such space. Although projecting does not work when $\mathbf{V}$ is not orthogonal, this idea can be integrated directly inside the Nguyen-Regev algorithm. More precisely, the Nguyen-Regev strategy can be summarised in three steps [NR09]:

1. find a linear transformation of $\mathbf{V}$ into an orthogonal matrix $\mathbf{A} = \mathbf{L}\mathbf{V}$;
2. recover a good approximation of a column $\pm\mathbf{A}$;

3. recover an approximation of a column of $\pm\mathbf{V}$ by multiplying by $\mathbf{L}^{-1}$.

When $\mathbf{A}$ is an orthogonal matrix, one can prove that its columns $\mathbf{a}_1, \ldots, \mathbf{a}_n$ are global minima on the unit sphere of $\mathbb{R}^n$ of a certain function. Hence, to obtain an approximation of all the columns of $\mathbf{V}$, we can simply modify the second step to return a full approximation of $\mathbf{A}$, by using projections. This gives the following corollary of Theorem 2.

Corollary 1. *There exists an algorithm $\mathcal{A}$ such that for any $c_0 > 0$, there exists $c_1 > 0$ such that given n^{c_1} samples uniformly distributed over some parallelepiped $\mathcal{P}_{[-1,1]}(\mathbf{V})$, for $\mathbf{V} \in \mathrm{GL}_n(\mathbb{R})$, the algorithm $\mathcal{A}$ returns with constant probability a matrix $\tilde{\mathbf{V}} = \mathbf{V}\tilde{\mathbf{E}}$, where each column of $\tilde{\mathbf{E}}$ is within ℓ_2 distance n^{-c_0} of a different standard basis vector $\pm\mathbf{e}_i$.*

4.3 Our algorithm based on statistical learning

In this section we describe our main algorithm for solving the hidden subset-sum problem. Recall that the first step of the original Nguyen-Stern algorithm returns a basis of the lattice $\bar{\mathcal{L}}_{\mathbf{x}}$. Given $\mathbf{X} \in \mathbb{Z}^{n \times m}$ the matrix of row vectors $\mathbf{x}_i \in \mathbb{Z}^m$, we obtain a matrix of row vectors $\mathbf{C} \in \mathbb{Z}^{n \times m}$ such that $\mathbf{C} = \mathbf{V}\mathbf{X}$ for $\mathbf{V} \in \mathrm{GL}_n(\mathbb{Q})$.

We observed that the columns of $\mathbf{C}$ can be interpreted as samples from a discrete parallelepiped $\mathcal{P}_{\{0,1\}}(\mathbf{V})$. As shown in Section 4.1, given a set $\mathfrak{C}$ of independent samples of the uniform distribution $\mathcal{P}_{\{0,1\}}(\mathbf{V})$, we can generate a set of independent samples $\mathfrak{V}$ of $\mathcal{P}_{[-1,1]}(\mathbf{V})$. Then, we can use $\mathfrak{V}$ as input for Nguyen-Regev algorithm to compute an approximation of $\mathbf{V}$, and eventually the $\mathbf{x}_i$'s can be recovered as $\mathbf{V}^{-1}\mathbf{C}$.

Algorithm 2 Statistical attack

Input: $\mathbf{h}, q, n$
Output: the vectors $\mathbf{x}_i$.
1: Compute a basis $\mathbf{C}$ of the lattice $\bar{\mathcal{L}}_{\mathbf{x}}$ by using Algorithm 1.
2: Generate a set of independent samples $\mathfrak{V}$ of $\mathcal{P}_{[-1,1]}(\mathbf{V})$ by using the m columns of $\mathbf{C}$ as shown in Section 4.1.
3: Use Nguyen-Regev algorithm to compute an approximation $\tilde{\mathbf{V}}$ of $\mathbf{V}$.
4: Recover $\mathbf{X}$ from the rounding of $\tilde{\mathbf{X}} = \tilde{\mathbf{V}}^{-1}\mathbf{C}$.
5: **return** $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$

Theorem 3. *There exist $n_0 \geq 0$ and $\ell > 0$ such that for any $n \geq n_0$, and for any prime integer q of bitsize at least $2mn\log(m)$, Algorithm 2 solves the hidden subset sum problem with constant probability in polynomial time, using $m = n^\ell$ samples.*

Proof. We show in Appendix A that the lattice $\bar{\mathcal{L}}_{\mathbf{x}}$ has rank n with constant probability. Therefore, we can apply Theorem 1 and obtain a basis $\mathbf{C}$ of $\bar{\mathcal{L}}_{\mathbf{x}}$ with constant probability. More precisely, we recover a matrix $\mathbf{C} \in \mathbb{Z}^{n \times m}$ such that $\mathbf{C} = \mathbf{V}\mathbf{X}$ for $\mathbf{V} \in \mathrm{GL}_n(\mathbb{Q})$. Moreover, we have $\mathbf{W}\mathbf{C} = \mathbf{X}$ where $\mathbf{W} = \mathbf{V}^{-1}$ and $\mathbf{W} \in \mathbb{Z}^{n \times n}$.

The columns of $\mathbf{C}$ can be interpreted as m samples from the discrete parallelepiped $\mathcal{P}_{\{0,1\}}(\mathbf{V})$. Lemma 2 implies that we can generate $m' = m/k$ samples from the continuous parallelepiped $\mathcal{P}_{[-1,1]}(\mathbf{V})$. Hence, we can apply Corollary 1 with $c_0 = 2$ and $m' = n^{c_1}$ samples from $\mathcal{P}_{[-1,1]}(\mathbf{V})$, and we can then recover a matrix $\tilde{\mathbf{V}}$ whose columns are close to the columns of $\pm\mathbf{V}$, with constant probability. Without loss of generality, we can assume that the columns of $\tilde{\mathbf{V}}$ are close to the columns of $\mathbf{V}$ (instead of $\pm\mathbf{V}$), by checking at Step 4 that the rows of $\mathbf{X}$ have components in $\{0, 1\}$ instead of $\{-1, 0\}$.

Namely, this implies that there exists a matrix $\tilde{\mathbf{E}}$ such that $\tilde{\mathbf{V}} = \mathbf{V}\tilde{\mathbf{E}}$ and each of its column is within ℓ_2 distance $\varepsilon = n^{-2}$ of some standard basis vector $\mathbf{e}_i$. Therefore, there exists a permutation matrix $\mathbf{P}$ such that $\tilde{\mathbf{E}}\mathbf{P}$ is close to the identity matrix, namely the respective columns have ℓ_2

distance smaller than ε. Without loss of generality, we can assume $\mathbf{P} = \mathbf{I}$. Thus, it holds true $\|\tilde{\mathbf{E}} - \mathbf{I}\|_{\max} < \varepsilon$, where we define $\|\mathbf{A}\|_{\max} \coloneqq \sup_{i,j} |a_{i,j}|$.

In order to recover the vectors $\mathbf{x}_i$'s by rounding the rows $\tilde{\mathbf{x}}_i$'s of $\tilde{\mathbf{X}} = \tilde{\mathbf{V}}^{-1}\mathbf{C}$, we must have $\|\mathbf{x}_i - \tilde{\mathbf{x}}_i\|_\infty < 1/2$. For any column $\mathbf{c}$ of $\mathbf{C}$ there exists a corresponding column $\mathbf{x} \in \{0,1\}^n$ of $\mathbf{X}$ such that $\mathbf{c} = \mathbf{V}\mathbf{x}$. So, if $\tilde{\mathbf{W}} = \tilde{\mathbf{V}}^{-1}$, then

$$\tilde{\mathbf{W}}\mathbf{c} - \mathbf{x} = \tilde{\mathbf{E}}^{-1}\mathbf{V}^{-1}\mathbf{V}\mathbf{x} - \mathbf{x} = (\tilde{\mathbf{E}}^{-1} - \mathbf{I})\mathbf{x}$$

This implies that $\|\tilde{\mathbf{W}}\mathbf{c} - \mathbf{x}\|_\infty \leq \|\tilde{\mathbf{E}}^{-1} - \mathbf{I}\|_\infty$. Therefore, a sufficient condition to recover the columns of $\mathbf{X}$ exactly is $\|\tilde{\mathbf{E}}^{-1} - \mathbf{I}\|_\infty < \frac{1}{2}$.

Given a matrix $\mathbf{Q}$ such that $\|\mathbf{Q}\|_\infty < 1$, then $(\mathbf{I} - \mathbf{Q})^{-1} = \sum_{j \geq 0} \mathbf{Q}^j$. Moreover, if $\|\mathbf{Q}\|_\infty < \alpha < 1/2$, we get:

$$\|(\mathbf{I} - \mathbf{Q})^{-1} - \mathbf{I}\|_\infty = \Big\| \sum_{j \geq 1} \mathbf{Q}^j \Big\|_\infty \leq \sum_{j=1}^{\infty} \alpha^j = \frac{\alpha}{1-\alpha} \leq 2\alpha$$

We can apply this inequality for $\mathbf{Q} = \mathbf{I} - \tilde{\mathbf{E}}$. Indeed, using $\|\tilde{\mathbf{E}} - \mathbf{I}\|_{\max} < \varepsilon$, we obtain $\|\mathbf{Q}\|_\infty \leq n \cdot \|\mathbf{Q}\|_{\max} < n\varepsilon \leq n^{-1}$. This gives $\|\tilde{\mathbf{E}}^{-1} - \mathbf{I}\|_\infty \leq 2n^{-1} \leq \frac{1}{2}$ as required.

We proved that $m' = \mathsf{poly}(n)$ samples of $\mathcal{U}(\mathcal{P}_{[-1,1]}(\mathbf{V}))$ are sufficient for applying Nguyen-Regev attack and computing the binary hidden vectors, and we observed that, from Lemma 2, those can be produced by $m = k \cdot m'$ samples from the hidden subset-sum problem. In addition, we notice that k can be choosen as polynomial in n, because both $\log \|\mathbf{V}\|_\infty$ and $\log \|\mathbf{V}^{-1}\|_\infty$ are polynomial in n. Indeed, if $\mathbf{C}_0$ is a $n \times n$ invertible submatrix of $\mathbf{C} \in \mathbb{Z}^{n \times m}$ and $\mathbf{X}_0$ the corresponding submatrix of $\mathbf{X} \in \{0,1\}^{n \times m}$, then we have $\mathbf{W} = \mathbf{V}^{-1} = \mathbf{X}_0\mathbf{C}_0^{-1}$ where the coefficients of both $\mathbf{C}_0$ and $\mathbf{X}_0$ have size polynomial in n; then $\log \|\mathbf{W}\|_\infty$ is polynomial in n. The same holds for the matrix $\mathbf{V} = \mathbf{W}^{-1}$. Hence, a number of samples m polynomial in n is enough, i.e. there exist $\ell > 0$ such that provided $m = n^\ell$ hidden subset sum samples we can solve the problem, within some constant probability.

Moreover, this implies that the time complexity of Algorithm 2 is polynomial in n. Indeed, every step is performed applying linear algebra operations on matrices of dimensions polynomial in n, and whose coefficients have size polynomial in n, too. More specifically, if v is the size of the coefficients of the elements in $\mathfrak{V}$, Nguyen-Regev algorithm's complexity is $\mathsf{poly}(n, m', v)$; see [NR09]. Moreover, the values of v and m' polynomially depend on n, k, m and the size of the coefficients of $\mathbf{C}$, which we already showed to be themselves polynomial in n.

5 The hidden linear combination problem

In this section we consider a natural generalisation of the hidden subset sum problem where the coefficients of the vectors $\mathbf{x}_i$'s lie in a discrete interval $[0, B] \cap \mathbb{Z}$ instead of $\{0,1\}$; we call this the *hidden linear combination problem*.

Definition 2 (Hidden Linear Combination Problem). *Let q be a positive integer, and let $\alpha_1, \ldots, \alpha_n$ be random integers in $\mathbb{Z}_q$. Let $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{Z}^m$ be random vectors with components in $[0, B] \cap \mathbb{Z}$. Let $\mathbf{h} = (h_1, \ldots, h_m) \in \mathbb{Z}^m$ satisfying:*

$$\mathbf{h} = \alpha_1\mathbf{x}_1 + \alpha_2\mathbf{x}_2 + \cdots + \alpha_n\mathbf{x}_n \pmod{q} \tag{4}$$

Given q and $\mathbf{h}$, recover the vector $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$ and the vectors $\mathbf{x}_i$'s, up to a permutation of the α_i's and $\mathbf{x}_i$'s.

We first describe an extension of the Nguyen-Stern algorithm for solving the above problem, with heuristic complexity exponential in n and polynomial in $\log B$. Namely, the orthogonal lattice attack in the first step recovers as previously the completion of the lattice $\mathcal{L}_{\mathbf{x}}$ generated by $\mathbf{x}_1, \ldots, \mathbf{x}_n$, and in the second step, one can still apply BKZ to recover the original vectors $\mathbf{x}_i$. Note that the

multivariate approach from [CG20] does not apply to the generalised problem, as it it would lead to multivariate polynomials of degree $B+1$. The technique would then remain polynomial-time only for constant B, and be essentially unpractical.

Secondly, we describe a statistical learning approach very close to Nguyen-Regev algorithm [NR09]. As opposed to the previous section we do not claim to have a proven algorithm: for the generalised problem, we only obtain a heuristic complexity $\mathsf{poly}(n, B)$.

5.1 Extending the Nguyen-Stern algorithm

The Nguyen-Stern algorithm can be adapted to the hidden linear combination problem; its heuristic complexity remains exponential in n as in the binary case, and polynomial in $\log B$. Recall that the algorithm has two steps:

1. From the samples $\mathbf{h}$, determine the lattice $\bar{\mathcal{L}}_{\mathbf{x}}$, where $\mathcal{L}_{\mathbf{x}}$ is the lattice generated by the $\mathbf{x}_i$'s.
2. From $\bar{\mathcal{L}}_{\mathbf{x}}$, recover the hidden vectors $\mathbf{x}_i$'s using BKZ. From $\mathbf{h}$, the $\mathbf{x}_i$'s and q, recover the weights α_i.

For the first step, the orthogonal lattice attack as recalled in Section 3.1 can be directly applied with $\mathbf{h}, q, n$ as input. The only difference is that the minimal size of the modulus q now depends on B. We prove the following generalisation of Theorem 1 in Appendix B.

Theorem 4. *Let $m > n$. Assume that the lattice $\mathcal{L}_{\mathbf{x}}$ has rank n. With probability at least $1/2$ over the choice of $\boldsymbol{\alpha}$, Algorithm 1 recovers a basis of $\bar{\mathcal{L}}_{\mathbf{x}}$ in polynomial time, assuming that q is a prime integer of bitsize at least $2mn\log m + (n+1)n\log B$. For $m = 2n$, the density is $d = n\log(B+1)/\log q = \mathcal{O}(1/(n\log n))$.*

The second step of Nguyen-Stern for the hidden linear combination problem is also essentially the same as for the hidden subset-sum problem. As previously, the first step of the Nguyen-Stern algorithm produces an LLL-reduced basis $(\mathbf{c}_1, \ldots, \mathbf{c}_n)$ of the completed lattice $\bar{\mathcal{L}}_{\mathbf{x}} \subset \mathbb{Z}^m$; however, the recovered basis vectors $(\mathbf{c}_1, \ldots, \mathbf{c}_n)$ can be much larger than the original vectors $\mathbf{x}_i$. Since we expect the $\mathbf{x}_i$'s to be among the shortest vectors in $\mathcal{L}_{\mathbf{x}}$, one must apply BKZ to obtain a better approximation factor. We provide a detailed analysis in Appendix B.2, following the analysis from [CG20]. We show that we obtain a similar condition on the BKZ approximation factor as in the binary case, and eventually the heuristic running time is $2^{\Omega(n/\log n)} \cdot \log^{\mathcal{O}(1)} B$. We provide in Section 6.1 the result of practical experiments.

5.2 Our statistical learning approach

In Section 4 we observed that a basis of the lattice $\bar{\mathcal{L}}_{\mathbf{x}}$ of the hidden subset sum problem can be interpreted as a set of samples of the uniform distribution over a discrete parallelepiped $\mathcal{P}_{\{0,1\}}(\mathbf{V})$. Similarly, in the hidden linear combination problem, the vector $\mathbf{x}_i$'s have coordinates uniformly distributed over $\{0, \ldots, B\}$. Given $\mathbf{X} \in \mathbb{Z}^{n\times m}$ the matrix of row vectors $\mathbf{x}_i$'s, $\mathbf{C} \in \mathbb{Z}^{n\times m}$ a basis of $\bar{\mathcal{L}}_{\mathbf{x}}$ and $\mathbf{V} \in \mathrm{GL}_n(\mathbb{Q})$ such that $\mathbf{C} = \mathbf{V}\cdot\mathbf{X}$, then the m columns of $\mathbf{C}$ are samples from the discrete parallelepiped $\mathcal{P}_{\{0,\ldots,B\}}(\mathbf{V}) = \{\mathbf{V}\mathbf{x} \colon \mathbf{x} \in \{0, \ldots, B\}^n\}$.

Instead of producing a continuous distribution and applying Nguyen-Regev as in Section 4 to recover an approximation of the matrix $\mathbf{V}$, it is more efficient in practice to apply the FastICA algorithm [HO97]; notice this also applies to the binary case with $B = 1$. Indeed, the Nguyen-Regev algorithm can be seen as an instantiation of the FastICA algorithm, i.e. an algorithm for solving the *signal source separation problem*, with kurtosis chosen as cost function.

Recall that the Nguyen-Regev statistical attack is composed of three steps:

1. find a linear transformation of $\mathbf{V}$ into an orthogonal matrix $\mathbf{A} = \mathbf{L}\mathbf{V}$;

2. recover a good approximation of a column $\pm\mathbf{A}$;
3. recover an approximation of a column of $\pm\mathbf{V}$ by multiplying by $\mathbf{L}^{-1}$.

This is actually the same overall strategy of FastICA. In both algorithms, the first step is performed by exploiting the sample covariance matrix leakage. For the second step, the Nguyen-Regev algorithm uses a gradient descent for minimising a certain function associated to $\mathcal{U}(\mathcal{P}_{[-1,1]}(\mathbf{A}))$. Actually, the choice of parameters in [NR09] makes such gradient descent coincide with the fixed-point algorithm used in FastICA, when the distribution of the sources is $\mathcal{U}([-1,1])$.

Therefore, our approach consists of applying directly FastICA to solve the hidden linear combination problem. Our algorithm works as follows:

1. compute a basis $\mathbf{C}$ of the lattice $\bar{\mathcal{L}}_{\mathbf{x}}$ using the orthogonal lattice attack;
2. use FastICA to compute an approximation $\tilde{\mathbf{V}}$ of $\mathbf{V}$.
3. compute $\mathbf{X}$ from the rounding of $\tilde{\mathbf{X}} = \tilde{\mathbf{V}}^{-1}\mathbf{C}$.

Here, we only provide a heuristic analysis of such algorithm. As in Nguyen-Regev, the FastICA algorithm produces an approximation $\tilde{\mathbf{V}}$ of $\mathbf{V}$, with relative distance n^{-c_0}, using $m = n^{c_1}$ samples. This gives an approximation $\tilde{\mathbf{X}}$ of $\mathbf{X}$ with relative distance $n^{-c'}$. Recall that the components of $\mathbf{X}$ are integers in $[0, B]$. Therefore, if $n^{-c'} < 1/(2B)$, we can recover $\mathbf{X}$ by rounding to the nearest integer. This shows that heuristically $\mathsf{poly}(n, B)$ samples are sufficient.

As the Nguyen-Regev algorithm, the complexity of FastICA is polynomial in the number of samples and the size of such vector's coefficients. Hence, the heuristic complexity of our statistical approach is $\mathsf{poly}(n, B)$. Recall that the Nguyen-Stern attack from Section 5.1 has complexity exponential in n but polynomial in $\log B$. We do not know how to solve the problem with complexity polynomial in both n and $\log B$.

The statistical learning approach based on FastICA is quite efficient in practice. For the hidden subset sum problem ($B = 1$), the approach requires $m \simeq n^2$ samples and has space complexity $\mathcal{O}(n^3)$ only, instead of $\mathcal{O}(n^4)$ in the multivariate approach from [CG20]; it is also much faster. Moreover, notice that the statistical approach described in this section does not require to transform the samples as in Step 2 of Algorithm 2. This is because applying FastICA allows us to manage discrete distributions, directly. We provide in Section 6.2 the result of practical experiments.

6 Practical experiments

We performed experiments for both the Nguyen-Stern and our statistical attack for the hidden linear combination problem when $B = 1$, i.e. when it coincides with the hidden subset sum problem, and $B = 10$. To facilitate the comparison, the modulus choice and the first step implementation follow the specifications of [CG20].

6.1 The Nguyen-Stern attack

We provide the result of practical experiments running the Nguyen-Stern algorithm in the binary case ($B = 1$) in Table 2, and for $B = 10$ in Table 3. For the binary case, we obtain similar timings as in [CG20]. In both cases we face an exponential barrier in the second step for $n > 170$.

			Step 1		Step 2			
n	m	$\log q$	LLL $\mathcal{L}_0$	LLL $\mathcal{L}_{\mathbf{x}}^{\perp}$	Hermite	Reduction		Total
70	140	772	3 s	1 s	1.021^n	LLL	1 s	6 s
90	180	1151	10 s	5 s	1.017^n	BKZ-10	1 s	19 s
110	220	1592	30 s	12 s	1.015^n	BKZ-10	3 s	50 s
130	260	2095	78 s	24 s	1.013^n	BKZ-20	12 s	123 s
150	300	2659	3 min	49 s	1.012^n	BKZ-30	135 s	7 min
170	340	3282	6 min	106 s	1.011^n	BKZ-30	260 min	269 min

Table 2. Running time of the Nguyen-Stern attack for $B = 1$ under a 3,2 GHz Intel Core i5 processor.

			Step 1		Step 2			
n	m	$\log q$	LLL $\mathcal{L}_0$	LLL $\mathcal{L}_{\mathbf{x}}^{\perp}$	Hermite	Reduction		Total
70	140	1237	7 s	4 s	1.021^n	LLL	1 s	12 s
90	180	1749	22 s	12 s	1.017^n	BKZ-10	1 s	37 s
110	220	2323	61 s	27 s	1.015^n	BKZ-10	2 s	96 s
130	260	2959	139 s	54 s	1.013^n	BKZ-20	8 s	4 min
150	300	3655	5 min	107 s	1.012^n	BKZ-30	3 min	12 min
170	340	4412	22 min	4 min	1.011^n	BKZ-30	139 min	167 min

Table 3. Running time of the Nguyen-Stern attack for $B = 10$ under a 3,2 GHz Intel Core i5 processor.

6.2 Statistical attack

We provide in Table 4 the results of practical experiments running our statistical attack for $B = 1$, and in Table 5 for $B = 10$. We have used the implementation of FastICA iterative algorithm provided in the `scikit-learn` package [PVG+11]. We see that for $B = 1$ the FastICA algorithm at Step 2 of the attack is very efficient, as we can reach $n = 250$, whereas with Nguyen-Stern we cannot solve the hidden subset-sum problem for $n > 170$. In particular, for $n = 250$, FastICA takes only 76 seconds, whereas in the multivariate attack from [CG20], the second step takes 45 minutes.

Although, for $B = 10$, in our experiments the statistical attack is less efficient than Nguyen-Stern, as one must generate a large number m of samples in the first step; the attack scales polynomially with n. Therefore, we expect the statistical approach to eventually outperform Nguyen-Stern for larger values of n. We provide the source code in:

https://pastebin.com/WzGXHmpW

n	m	$\log q$	Step 1: LLL	Step 2: FastICA	Total
70	4900	986	13 s	2 s	17 s
90	8100	1443	40 s	2 s	45 s
110	12100	1965	96 s	4 s	106 s
130	16900	2552	3 min	7 s	5 min
150	22500	3201	8 min	11 s	9 min
170	28900	3912	15 min	15 s	17 min
190	36100	4684	32 min	25 s	33 min
220	48400	5955	128 min	39 s	130 min
250	62500	7362	230 min	76 s	233 min

Table 4. Running time of our statistical attack for $B = 1$ under a 3,2 GHz Intel Core i5 processor.

n	m	$\log q$	Step 1: LLL	Step 2: FastICA	Total
70	147000	1237	4 min	27 s	6 min
90	189000	1749	9 min	50 s	12 min
110	231000	2323	17 min	71 s	21 min
130	273000	2959	31 min	98 s	36 min
150	315000	3655	66 min	139 s	73 min

Table 5. Running time of our statistical attack for $B = 10$ under a 3,2 GHz Intel Core i5 processor.

7 Conclusion

We have described a proven polynomial-time algorithm for solving the hidden subset-sum problem, based on the Nguyen-Stern orthogonal lattice attack [NS99], and on the Nguyen-Regev statistical learning attack [NR09]. The original Nguyen-Stern algorithm for the hidden subset-sum problem has exponential complexity, while the multivariate attack in [CG20] is heuristic polynomial-time only. We have also considered a natural generalisation of the hidden subset sum, with integer coefficients uniformly distributed between 0 and B instead of binary. In that case the multivariate approach from [CG20] does not apply, but our statistical approach can still be extended, at least heuristically, to get polynomial-time complexity in both n and B.

Our proven polynomial-time algorithm for solving the hidden subset-sum problem is of theoretical interest only, as it would require a huge number of samples. For our practical experiments, we have used the FastICA algorithm, and for the hidden subset-sum problem, we obtained at least an order of magnitude improvement in running time for the second step, compared to the multivariate attack from [CG20], using a similar number of samples.

References

[BPV98] Victor Boyko, Marcus Peinado, and Ramarathnam Venkatesan. Speeding up discrete log and factoring based schemes via precomputations. In Kaisa Nyberg, editor, *Advances in Cryptology — EUROCRYPT'98*, pages 221–235, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[CG20] Jean-Sébastien Coron and Agnese Gini. A polynomial-time algorithm for solving the hidden subset sum problem. In *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part II*, 2020. Full version available at https://eprint.iacr.org/2020/461.

[CJL+92] Matthijs J. Coster, Antoine Joux, Brian A. LaMacchia, Andrew M. Odlyzko, Claus-Peter Schnorr, and Jacques Stern. Improved low-density subset sum algorithms. *Computational Complexity*, 2:111–128, 1992.

[CN11] Yuanmi Chen and Phong Q. Nguyen. BKZ 2.0: Better lattice security estimates. In *Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings*, pages 1–20, 2011.

[HO97] Aapo Hyvärinen and Erkki Oja. A fast fixed-point algorithm for independent component analysis. *Neural Computation*, 9(7):1483–1492, 1997.

[HPS11] Guillaume Hanrot, Xavier Pujol, and Damien Stehlé. Analyzing blockwise lattice algorithms using dynamical systems. In *CRYPTO 2011*, 2011.

[LLL82] Arjen K. Lenstra, Hendrik W. Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *MATH. ANN*, 261:515–534, 1982.

[LO85] Jeffrey C. Lagarias and Andrew M. Odlyzko. Solving low-density subset sum problems. *J. ACM*, 32(1):229–246, 1985.

[NR09] Phong Q. Nguyen and Oded Regev. Learning a parallelepiped: Cryptanalysis of GGH and NTRU signatures. *J. Cryptology*, 22(2):139–160, 2009.

[NS99] Phong Q. Nguyen and Jacques Stern. The hardness of the hidden subset sum problem and its cryptographic implications. In *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, pages 31–46, 1999.

[NS09] Phong Q. Nguyen and Damien Stehlé. An LLL algorithm with quadratic complexity. *SIAM J. Comput.*, 39(3):874–903, August 2009.

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[Sch87] Claus-Peter Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theor. Comput. Sci.*, 53:201–224, 1987.

A Rank of $\mathcal{L}_{\mathbf{x}}$

The lattice $\mathcal{L}_{\mathbf{x}}$ is defined as the lattice generated by $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \{0,1\}^m$ for $m \geq n$. The probability that $\mathcal{L}_{\mathbf{x}}$ has full rank n is lower bounded by the probability that a random $n \times n$ binary matrix is invertible in $\mathbb{F}_2$. Let $p(n)$ be this probability. We have:

$$p(n) = 2^{-n^2} \cdot \prod_{k=1}^{n} \left(2^n - 2^{k-1}\right) = \prod_{i=1}^{n} \left(1 - 2^{-i}\right)$$

Namely, the first row must be non-zero, so there are $2^n - 1$ possibilities, and for $2 \leq k \leq n$ the k-th row must be linearly independent from the first $k-1$ rows, so there are $2^n - 2^{k-1}$ possibilities.

Moreover, using $1 - x \geq \exp(-2x)$ for $0 \leq x \leq 1/2$, we obtain:

$$p(n) \geq \prod_{k=1}^{n} \exp(-2 \cdot 2^{-k}) = \exp\left(\sum_{k=0}^{n-1} 2^{-i}\right) \geq e^{-2}$$

Therefore the lattice $\mathcal{L}_{\mathbf{x}}$ has full rank n with at least constant probability.

B The Nguyen-Stern algorithm

Nguyen and Stern in [NS99] present a lattice based algorithm for solving the hidden subset sum algorithm. In this section we describe a straightforward generalisation of the Nguyen-Stern algorithm for solving the hidden linear combination problem. We describe this attack following the analysis of the Nguyen-Stern algorithm of [CG20].

Recall that in the hidden combination problem, given a modulus q and $\mathbf{h} = (h_1, \ldots, h_m) \in \mathbb{Z}^m$ satisfying

$$\mathbf{h} = \alpha_1 \mathbf{x}_1 + \alpha_2 \mathbf{x}_2 + \cdots + \alpha_n \mathbf{x}_n \pmod{q} \tag{5}$$

we must recover the vector $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n) \in \mathbb{Z}_q^n$ and the vectors $\mathbf{x}_i \in ([0, B] \cap \mathbb{Z})^m$. The hidden subset-sum problem corresponds to $B = 1$. The Nguyen-Stern algorithm comprises two steps:

1. From the samples $\mathbf{h}$, determine the lattice $\bar{\mathcal{L}}_{\mathbf{x}}$, where $\mathcal{L}_{\mathbf{x}}$ is the lattice generated by the $\mathbf{x}_i$'s.
2. From $\bar{\mathcal{L}}_{\mathbf{x}}$, recover the hidden vectors $\mathbf{x}_i$'s using BKZ. From $\mathbf{h}$, the $\mathbf{x}_i$'s and q, recover the weights α_i.

B.1 First step: orthogonal lattice attack

In the first step the goal is to recover $\bar{\mathcal{L}}_{\mathbf{x}}$ from $\mathbf{h}$ and q. Let $\mathcal{L}_0$ be the lattice of vectors orthogonal to $\mathbf{h}$ modulo q:

$$\mathcal{L}_0 \coloneqq \Lambda_q^{\perp}(\mathbf{h}) = \{\mathbf{u} \in \mathbb{Z}^m \mid \langle \mathbf{u}, \mathbf{h} \rangle \equiv 0 \pmod{q}\}$$

For any $\mathbf{u} \in \mathcal{L}_0$, the vector

$$\mathbf{p}_{\mathbf{u}} = (\langle \mathbf{u}, \mathbf{x}_1 \rangle, \ldots, \langle \mathbf{u}, \mathbf{x}_n \rangle)$$

is orthogonal to the vector $\boldsymbol{\alpha}$ modulo q, since from (5) we obtain:

$$\langle \mathbf{u}, \mathbf{h} \rangle \equiv \alpha_1 \langle \mathbf{u}, \mathbf{x}_1 \rangle + \cdots + \alpha_n \langle \mathbf{u}, \mathbf{x}_n \rangle \equiv 0 \pmod{q}.$$

Then if $\mathbf{p}_{\mathbf{u}}$ is shorter than the shortest non-zero vector orthogonal to $\boldsymbol{\alpha}$ modulo q, we must have $\mathbf{p}_{\mathbf{u}} = 0$, and therefore $\mathbf{u} \in \mathcal{L}_{\mathbf{x}}^{\perp}$.

Therefore, the orthogonal lattice attack first obtains an LLL-reduced basis of $\mathcal{L}_0$, from which it extracts a generating set for $\mathcal{L}_{\mathbf{x}}^{\perp}$; subsequently, it computes the orthogonal of $\mathcal{L}_{\mathbf{x}}^{\perp}$ obtaining $\bar{\mathcal{L}}_{\mathbf{x}}$. In the following we extend the analysis of [CG20] to $B > 1$.

Algorithm 3 Orthogonal lattice attack

Input: $\mathbf{h}, q, n, B$.
Output: A basis of $\bar{\mathcal{L}}_{\mathbf{x}}$.
1: Compute an LLL-reduced basis $\mathbf{u}_1, \ldots, \mathbf{u}_m$ of $\mathcal{L}_0$.
2: Extract a generating set of $\mathbf{u}_1, \ldots, \mathbf{u}_{m-n}$ of $\mathcal{L}_{\mathbf{x}}^{\perp}$.
3: Compute a basis $(\mathbf{c}_1, \ldots, \mathbf{c}_n)$ of $\bar{\mathcal{L}}_{\mathbf{x}} = (\mathcal{L}_{\mathbf{x}}^{\perp})^{\perp}$.
4: **return** $(\mathbf{c}_1, \ldots, \mathbf{c}_n)$

The orthogonal lattice attack described in Algorithm 3 is guaranteed to succeed with good probability for sufficiently large q. The following theorem is a generalisation of [CG20, Theorem 1].

Theorem 5 (Theorem 4). *Let $m > n$. Assume that the lattice $\mathcal{L}_{\mathbf{x}}$ has rank n. With probability at least $1/2$ over the choice of $\boldsymbol{\alpha}$, Algorithm 3 recovers a basis of $\bar{\mathcal{L}}_{\mathbf{x}}$ in polynomial time, assuming that q is a prime integer of bitsize at least $2mn \log m + (n+1)n \log B$. For $m = 2n$, the density is $d = n \log(B+1)/\log q = \mathcal{O}(1/(n \log n))$.*

We denote by $\Lambda_q^{\perp}(\boldsymbol{\alpha})$ the lattice of vectors orthogonal to $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$ modulo q. The proof of Theorem 5 is based on the following two lemmas:

Lemma 3. *Assume that the lattice $\mathcal{L}_{\mathbf{x}}$ has rank n. Algorithm 3 computes a basis of the lattice $\bar{\mathcal{L}}_{\mathbf{x}}$ in polynomial time under the condition $m > n$ and*

$$\sqrt{mn} \cdot B \cdot 2^{\frac{m}{2}} \cdot \lambda_{m-n}\left(\mathcal{L}_{\mathbf{x}}^{\perp}\right) < \lambda_1\left(\Lambda_q^{\perp}(\boldsymbol{\alpha})\right). \tag{6}$$

Proof. As observed previously, for any $\mathbf{u} \in \mathcal{L}_0$, the vector

$$\mathbf{p}_{\mathbf{u}} = (\langle \mathbf{u}, \mathbf{x}_1 \rangle, \ldots, \langle \mathbf{u}, \mathbf{x}_n \rangle)$$

is orthogonal to the vector $\boldsymbol{\alpha}$ modulo q, i.e. $\mathbf{p}_{\mathbf{u}} \in \Lambda_q^{\perp}(\boldsymbol{\alpha})$. Therefore, if $\mathbf{p}_{\mathbf{u}}$ is shorter than the shortest non-zero vector orthogonal to $\boldsymbol{\alpha}$ modulo q, we must have $\mathbf{p}_{\mathbf{u}} = 0$, and consequently $\mathbf{u} \in \mathcal{L}_{\mathbf{x}}^{\perp}$. Thus, a sufficient condition for $\mathbf{u} \in \mathcal{L}_{\mathbf{x}}^{\perp}$ is $\|\mathbf{p}_{\mathbf{u}}\| < \lambda_1\left(\Lambda_q^{\perp}(\boldsymbol{\alpha})\right)$.

Since $\|\mathbf{p}_{\mathbf{u}}\| \leq \sqrt{mn} B \|\mathbf{u}\|$, this implies that given any $\mathbf{u} \in \mathcal{L}_0$ we must have $\mathbf{u} \in \mathcal{L}_{\mathbf{x}}^{\perp}$ if

$$\sqrt{mn} B \|\mathbf{u}\| < \lambda_1\left(\Lambda_q^{\perp}(\boldsymbol{\alpha})\right). \tag{7}$$

Now, the lattice $\mathcal{L}_0$ is full rank of dimension m since it contains $q\mathbb{Z}^m$. Therefore, we can consider $\mathbf{u}_1, \ldots, \mathbf{u}_m$ an LLL-reduced basis of $\mathcal{L}_0$. From Lemma 1, for each $j \leq m - n$ we have

$$\|\mathbf{u}_j\| \leq 2^{\frac{m}{2}} \cdot \lambda_{m-n}(\mathcal{L}_0) \leq 2^{\frac{m}{2}} \cdot \lambda_{m-n}\left(\mathcal{L}_{\mathbf{x}}^{\perp}\right) \tag{8}$$

since $\mathcal{L}_{\mathbf{x}}^{\perp}$ is a sublattice of $\mathcal{L}_0$ of dimension $m - n$. Combining (8) with (7), when

$$\sqrt{mn} \cdot B \cdot 2^{\frac{m}{2}} \cdot \lambda_{m-n}\left(\mathcal{L}_{\mathbf{x}}^{\perp}\right) < \lambda_1\left(\Lambda_q^{\perp}(\boldsymbol{\alpha})\right)$$

the vectors $\mathbf{u}_1, \ldots, \mathbf{u}_{m-n}$ must belong to $\mathcal{L}_{\mathbf{x}}^{\perp}$. This means that $\langle \mathbf{u}_1, \ldots, \mathbf{u}_{m-n} \rangle$ is a full rank sublattice of $\mathcal{L}_{\mathbf{x}}^{\perp}$, and therefore $\langle \mathbf{u}_1, \ldots, \mathbf{u}_{m-n} \rangle^{\perp} = \bar{\mathcal{L}}_{\mathbf{x}}$. Finally, Algorithm 3 is polynomial-time, because both the LLL reduction step of $\mathcal{L}_0$ and the LLL-based orthogonal computation of $\mathcal{L}_{\mathbf{x}}^{\perp}$ are polynomial-time.

We recall [CG20, Lemma 3]:

Lemma 4. *Let q be a prime. Then with probability at least $1/2$ over the choice of $\boldsymbol{\alpha}$, we have $\lambda_1(\Lambda_q^{\perp}(\boldsymbol{\alpha})) \geq q^{1/n}/4$.*

Proof (Proof of Theorem 5). In order to apply Lemma 3, we first derive an upper-bound on $\lambda_{m-n}\left(\mathcal{L}_{\mathbf{x}}^{\perp}\right)$. The lattice $\mathcal{L}_{\mathbf{x}}^{\perp}$ has dimension $m-n$, therefore by Minkowski's second theorem we have

$$\lambda_{m-n}\left(\mathcal{L}_{\mathbf{x}}^{\perp}\right) \leq \sqrt{\gamma_{m-n}}^{m-n} \det\left(\mathcal{L}_{\mathbf{x}}^{\perp}\right) \leq m^{m/2} \det\left(\mathcal{L}_{\mathbf{x}}^{\perp}\right). \tag{9}$$

From $\det \mathcal{L}_{\mathbf{x}}^{\perp} = \det \bar{\mathcal{L}}_{\mathbf{x}} \leq \det \mathcal{L}_{\mathbf{x}}$ and Hadamard's inequality with $\|\mathbf{x}_i\| \leq B\sqrt{m}$, we obtain:

$$\det \mathcal{L}_{\mathbf{x}}^{\perp} \leq \det \mathcal{L}_{\mathbf{x}} \leq \prod_{i=1}^{n} \|\mathbf{x}_i\| \leq m^{n/2} B^n \tag{10}$$

which gives the following upper-bound on $\lambda_{m-n}\left(\mathcal{L}_{\mathbf{x}}^{\perp}\right)$:

$$\lambda_{m-n}\left(\mathcal{L}_{\mathbf{x}}^{\perp}\right) \leq m^{m/2} m^{n/2} B^n \leq m^m B^n.$$

Thus, by Lemma 3, we can recover a basis of $\bar{\mathcal{L}}_{\mathbf{x}}$ when

$$\sqrt{mn} \cdot 2^{\frac{m}{2}} \cdot m^m \cdot B^{n+1} < \lambda_1\left(\Lambda_q^{\perp}(\boldsymbol{\alpha})\right).$$

By Lemma 4, this implies that, with probability at least $1/2$ over the choice of $\boldsymbol{\alpha}$, we can recover the hidden lattice $\bar{\mathcal{L}}_{\mathbf{x}}$ if:

$$\sqrt{mn} \cdot 2^{\frac{m}{2}} \cdot m^m \cdot B^{n+1} < q^{1/n}/4.$$

For $m > n \geq 4$, therefore it suffices to have $\log q \geq 2mn \log m + (n+1)n \log B$.

Heuristic analysis. In practice we can use a smaller value for the modulus q than predicted by Theorem 5. As in [CG20], we derive a heuristic size for the modulus q, using an approximation of the terms in condition (6). We start with the term $\lambda_{m-n}\left(\mathcal{L}_{\mathbf{x}}^{\perp}\right)$. For a "random lattice" we expect the lattice minima to be balanced, and therefore $\lambda_{m-n}\left(\mathcal{L}_{\mathbf{x}}^{\perp}\right)$ to be roughly equal to $\lambda_1\left(\mathcal{L}_{\mathbf{x}}^{\perp}\right)$. Therefore we use the heuristic approximation:

$$\lambda_{m-n}\left(\mathcal{L}_{\mathbf{x}}^{\perp}\right) \simeq \sqrt{\gamma_{m-n}} \det(\mathcal{L}_{\mathbf{x}}^{\perp})^{\frac{1}{m-n}}.$$

From (10) we obtain:

$$\lambda_{m-n}\left(\mathcal{L}_{\mathbf{x}}^{\perp}\right) \lessapprox \sqrt{\gamma_{m-n}} B^{\frac{n}{m-n}} m^{\frac{n}{2(m-n)}}. \tag{11}$$

For the term $\lambda_1\left(\Lambda_q^{\perp}(\boldsymbol{\alpha})\right)$, using the Gaussian heuristic, we expect:

$$\lambda_1\left(\Lambda_q^{\perp}(\boldsymbol{\alpha})\right) \simeq \sqrt{\gamma_n} q^{\frac{1}{n}}.$$

Finally the $2^{m/2}$ factor in (6) corresponds to the LLL Hermite factor with $\delta = 3/4$; in practice we will use $\delta = 0.99$, and we denote by $2^{\iota m}$ the corresponding LLL Hermite factor. Hence from (6) we obtain the heuristic condition:

$$\sqrt{mn} \cdot B \cdot 2^{\iota \cdot m} \cdot \sqrt{\gamma_{m-n}} \cdot B^{\frac{n}{m-n}} \cdot m^{\frac{n}{2(m-n)}} < \sqrt{\gamma_n} q^{1/n}.$$

This gives the condition:

$$2^{\iota \cdot m} \sqrt{\gamma_{m-n} \cdot n} \cdot B^{\frac{m}{m-n}} \cdot m^{\frac{m}{2(m-n)}} < \sqrt{\gamma_n} q^{1/n}$$

which gives:

$$\log q > \iota \cdot m \cdot n + \frac{n}{2} \log(n \cdot \gamma_{m-n}/\gamma_n) + \frac{mn}{2(m-n)} \log m + \frac{n \cdot m}{m-n} \log B \tag{12}$$

For $m = 2n$ we obtain the condition:

$$\log q > 2\iota \cdot n^2 + \frac{3n}{2} \log n + n + 2n \log B \tag{13}$$

In practice for our experiments we can use $m = 2n$ and $\log q \simeq 2\iota n^2 + n \log n + 2n \log B$ with $\iota = 0.035$.

B.2 Extended Nguyen-Stern attack: second step

The vectors $\mathbf{x}_i \in \mathbb{Z}^m$ have coordinates distributed uniformly over the set $\{0, \dots, B\}$. Then the expected value of $\|\mathbf{x}_i\|^2$ is $m \cdot \mathbb{E}[x^2] = m \cdot \mu'_{2,B} = m \cdot B(2B+1)/6$ for x any coordinate. This implies, by Jensen inequality, that we expect the norm of the $\mathbf{x}_i$'s to be smaller than $\sqrt{mB(2B+1)/6}$. In addition, the expected value of the norm of the difference between some $\mathbf{x}_i$ and $\mathbf{x}_j$ is

$$\mathbb{E}[\|\mathbf{x}_i - \mathbf{x}_j\|] \leq \sqrt{m \cdot 2\,\mathrm{Var}(x)} = \sqrt{m \cdot 2\sigma_B^2} = \sqrt{m\frac{B(B+2)}{6}}.$$

Notice that these bounds on the $\mathbf{x}_i$ and $\mathbf{x}_i - \mathbf{x}_j$ for $i \neq j$ coincide when $B = 1$. In general, we have that both these classes of vectors are short vectors of $\mathcal{L}_{\mathbf{x}}$.

Hence, after BKZ reduction of the first step's output basis $\mathbf{C}$ with a large enough block-size β, we expect that each vector of the basis vectors $\mathbf{c}_1, \dots, \mathbf{c}_n$ is either equal to $\pm\mathbf{x}_i$, or equal to a combination of the form $\mathbf{x}_i - \mathbf{x}_j$ for $i \neq j$.

Therefore, first we can collect in a set L the $\pm\mathbf{c}_i$'s whose coefficients are between 0 and B, i.e. the set of candidate target vectors. Notice that $L \neq \varnothing$ since at least one the $\mathbf{c}_i$ must be equal to one of the $\pm\mathbf{x}_j$'s; this is because, otherwise, the vector of all ones would belong to the kernel of the transition matrix $\mathbf{M}$ between $(\mathbf{x}_1, \dots, \mathbf{x}_n)$ and $(\mathbf{c}_1, \dots, \mathbf{c}_n)$, i.e. the latter would not be a basis. Without loss of generality, we can suppose $L = \{\mathbf{c}_1, \dots, \mathbf{c}_k\}$ and

$$\mathbf{M} = \left(\begin{array}{c|c} \mathbf{I}_k & \mathbf{0} \\ \hline \mathbf{A}_k & \mathbf{M}_{n-k} \end{array}\right)$$

If $k < n$, for $j = k+1, \dots, n$ and for any $\mathbf{v}$ in L we compute $\mathbf{c} = \pm\mathbf{c}_j - \mathbf{v}$, and check if either $\mathbf{c}$ or $-\mathbf{c}$ are suitable to be added to the set L. Again, we must find at least a new vector, otherwise the matrix $\mathbf{M}_{n-k}$ would be singular. Hence, iterating this reasoning and updating L at each step, we recover all the rows of $\mathbf{X}$.

The number of controls for round is $\mathcal{O}(n - \#L)$. Therefore, while staying in the lattice $\mathcal{L}_{\mathbf{x}}$ we can recover each of the original vectors $\mathbf{x}_i$ from the basis vectors $\mathbf{C}$, by $\mathcal{O}(n^3)$ tests.

BKZ block-size and running time. We can construct a "generic" short vector in $\mathcal{L}_{\mathbf{x}}$ as a binary combination of vectors of the form $\mathbf{x}_i - \mathbf{x}_j$. Namely, consider $\mathbf{z} = \sum_{k=1}^{n} b_k \mathbf{z}_k$ with $b_j \in \{0, 1\}$ and $\mathbf{z}_k$ a $\mathbf{x}_i - \mathbf{x}_j$. The variance of any component of $\mathbf{z}$ is

$$\mathrm{Var}(z_i) = \sum_k \mathrm{Var}(b_j)\,\mathrm{Var}(z_{ki}) = \frac{n}{2}\sigma_B^2,$$

where σ_B^2 is the variance of the uniform distribution over $\{0, \dots, B\}$. Thus the norm of the resulting vector will be about $\sqrt{mn\sigma_B^2/2}$.

Then heuristically the gap between these generic vectors and the shortest vectors is:

$$\frac{\sqrt{mnB(B+2)/24}}{\sqrt{mB(2B+1)/6}} = \frac{1}{2}\sqrt{n\frac{B+2}{2B+1}}.$$

Therefore, in order to recover the shortest vectors, the BKZ approximation factor $2^{\iota \cdot n}$ should be less than such gap, namely:

$$2^{\iota \cdot n} \leq \frac{1}{2}\sqrt{n\frac{B+2}{2B+1}} \simeq \frac{1}{2}\sqrt{\frac{n}{2}} \tag{14}$$

which gives $\iota \leq (\log(n/8))/(2n)$. Achieving an Hermite factor of $2^{\iota n}$ heuristically requires at least $2^{\Omega(1/\iota)}$ time, by using BKZ reduction with block-size $\beta = \omega(1/\iota)$ [HPS11]. This implies that to satisfy (14), we should consider $\beta = \omega(n/\log n)$. Namely, the running time of the generalised Nguyen-Stern algorithm results $\mathsf{poly}(n, \log B) \cdot 2^{\Omega(n/\log n)} = 2^{\Omega(n/\log n)} \cdot \log^{\mathcal{O}(1)} B$.