A Polynomial-Time Algorithm for Solving the Hidden Subset Sum Problem

Jean-Sébastien Coron$^{(\boxtimes)}$ and Agnese Gini$^{(\boxtimes)}$

University of Luxembourg, Esch-sur-Alzette, Luxembourg
`jean-sebastien.coron@uni.lu`, `agnese.gini@uni.lu`

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. While the Nguyen-Stern algorithm works quite well in practice for moderate values of n, we argue that its complexity is actually exponential in n; namely in the final step one must recover a very short basis of a n-dimensional lattice, which takes exponential-time in n, as one must apply BKZ reduction with increasingly large block-sizes.

   In this paper, we describe a variant of the Nguyen-Stern algorithm that works in polynomial-time. The first step is the same orthogonal lattice attack with LLL as in the original algorithm. In the second step, instead of applying BKZ, we use a multivariate technique that recovers the short lattice vectors and finally the hidden secrets in polynomial time. Our algorithm works quite well in practice, as we can reach $n \simeq 250$ in a few hours on a single PC.

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 a 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 M be an integer, and let $\alpha_1, \ldots, \alpha_n$ be random integers in $\mathbb{Z}_M$. 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:*

$$\mathbf{h} = \alpha_1 \mathbf{x}_1 + \alpha_2 \mathbf{x}_2 + \cdots + \alpha_n \mathbf{x}_n \pmod{M} \qquad (1)$$

Given M 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 $d = n / \log M$ 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 $h, \alpha_1, \ldots, \alpha_n, M$; 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. This technique was introduced by Nguyen and Stern at Crypto '97 for breaking the Qu-Vanstone cryptosystem [NS97], and it has numerous applications in cryptanalysis, for example cryptanalysis of the Ajtai-Dwork cryptosystem [NS98b], cryptanalysis of the Béguin-Quisquater server-aided RSA protocol [NS98a], fault attacks against RSA-CRT signatures [CNT10, BNNT11], attacks against discrete-log based signature schemes [NSS04], and cryptanalysis of various homomorphic encryption schemes [vDGHV10, LT15, FLLT15] and multilinear maps [CLT13, CP19, CN19].

The orthogonal lattice attack against the hidden subset sum problem is based on the following technique [NS99]. If a vector $\mathbf{u}$ is orthogonal modulo M to the public vector of samples $\mathbf{h}$, then from (1) we must have:

$$\langle \mathbf{u}, \mathbf{h} \rangle \equiv \alpha_1 \langle \mathbf{u}, \mathbf{x_1} \rangle + \cdots + \alpha_n \langle \mathbf{u}, \mathbf{x_n} \rangle \equiv 0 \pmod{M}$$

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 M. 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 M, 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.

Complexity of the Nguyen-Stern algorithm. While the Nguyen-Stern algorithm works quite well in practice for moderate values of n, we argue that its complexity is actually exponential in the number of weights n. Namely in the first step we only recover 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 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 $\mathcal{L}_\mathbf{x}$, and in principle this takes exponential-time in n, as one must apply BKZ reduction [Sch87] with increasingly large block-sizes. In their practical experiments, the authors of [NS99] were able to solve the hidden subset sum problem up to $n = 90$; for the second step, they used a BKZ implementation from the NTL library [Sho] with block-size $\beta = 20$. In our implementation of

their algorithm, with more computing power and thanks to the BKZ 2.0 [CN11] implementation from [fpl16], we can reach $n = 170$ with block-size $\beta = 30$, but we face an exponential barrier beyond this value.

Our contributions. Our first contribution is to provide a more detailed analysis of both steps of the Nguyen-Stern algorithm. For the first step (orthogonal lattice attack with LLL), we first adapt the analysis of [NS99] to provide a rigorous condition under which the hidden lattice $\mathcal{L}_\mathbf{x}$ can be recovered. In particular, we derive a rigorous lower bound for the bitsize of the modulus M; we show that the knapsack density $d = n/\log M$ must be $\mathcal{O}(1/(n\log n))$, and heuristically $\mathcal{O}(1/n)$, as for the classical subset-sum problem.

We also provide a heuristic analysis of the second step of Nguyen-Stern. More precisely, we provide a simple model for the minimal BKZ block-size β that can recover the secret vectors $\mathbf{x}_i$, based on the gap between the shortest vectors and the other vectors of the lattice. While relatively simplistic, our model seems to accurately predict the minimal block-size β required for BKZ reduction in the second step. We show that under our model the BKZ block-size must grow almost linearly with the dimension n; therefore the complexity of the second step is exponential in n. We also provide a slightly simpler approach for recovering the hidden vectors $\mathbf{x}_i$ from the shortest lattice vectors. Eventually we argue that the asymptotic complexity of the full Nguyen-Stern algorithm is $2^{\Omega(n/\log n)}$.

Our main contribution is then to describe a variant of the Nguyen-Stern algorithm for solving the hidden subset sum problem that works in polynomial-time. The first step is still the same orthogonal lattice attack with LLL. In the second step, instead of applying BKZ, we use a multivariate technique that recovers the short lattice vectors and finally the hidden secrets in polynomial time, using $m \simeq n^2/2$ samples instead of $m = 2n$ as in [NS99]. Our new second step can be of independent interest, as its shows how to recover binary vectors in a lattice of high-dimensional vectors. Asymptotically the heuristic complexity of our full algorithm is $\mathcal{O}(n^9)$. We show that our algorithm performs quite well in practice, as we can reach $n \simeq 250$ in a few hours on a single PC.

Cryptographic applications. 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. Such generator can be used to speed-up the generation of discrete-log based algorithms with fixed base g, such as Schnorr identification, and Schnorr, ElGamal and DSS signatures. We show that in practice our polynomial-time algorithm enables to break the Boyko *et al.* generator for values of n that are beyond reach for the original Nguyen-Stern attack; however, we need more samples from the generator, namely $m \simeq n^2/2$ samples instead of $m = 2n$.

Source code. We provide in

the source code of the Nguyen-Stern attack and our new attack in `SageMath` [Sag19], using the L^2 [NS09] implementation from [fpl16].

2   Background on Lattices

Lattices and bases. In this section we recall the main definitions and properties of lattices used throughout this paper; we refer to the full version of this paper [CG20] for more details. Let $\mathbf{b}_1, \ldots, \mathbf{b}_d \in \mathbb{Z}^m$ be linearly independent vectors. The *lattice* generated by the basis $\mathbf{b}_1, \ldots, \mathbf{b}_d$ is the set

$$\mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_d) = \left\{ \sum_{i=1}^{d} a_i \mathbf{b}_i \mid a_1, \ldots, a_d \in \mathbb{Z} \right\}.$$

We say that a matrix $\mathbf{B}$ is a *base matrix* for the lattice generated by its rows $\mathbf{b}_1, \ldots, \mathbf{b}_d$. Two basis $\mathbf{B}, \mathbf{B}'$ generate the same lattice if and only if there exists an unimodular matrix $\mathbf{U} \in \mathrm{GL}(\mathbb{Z}, d)$ such that $\mathbf{UB} = \mathbf{B}'$. Given any basis $\mathbf{B}$ we can consider its Gram-determinant $d(\mathbf{B}) = \sqrt{\det(\mathbf{BB^\intercal})}$; this number is invariant under base change. The *determinant* of a lattice $\mathcal{L}$ is the Gram-determinant of any of its basis $\mathbf{B}$, namely $\det(\mathcal{L}) = d(\mathbf{B})$.

The *dimension* $\dim(\mathcal{L})$, or *rank*, of a lattice is the dimension as vector space of $E_{\mathcal{L}} := \mathrm{Span}_{\mathbb{R}}(\mathcal{L})$, namely the cardinality of its bases. We say that a lattice is *full rank* if it has maximal dimension. We say that $\mathcal{M} \subseteq \mathcal{L}$ is a *sublattice* of a lattice $\mathcal{L}$ if it is a lattice contained in $\mathcal{L}$, further we say that $\mathcal{L}$ is a *superlattice* of $\mathcal{M}$. If $\dim(\mathcal{M}) = \dim(\mathcal{L})$, we say that $\mathcal{M}$ is a full-rank sublattice of $\mathcal{L}$, and we must have $\det(\mathcal{L}) \leq \det(\mathcal{M})$.

Orthogonal lattice. Consider the Euclidean norm $\|\cdot\|$ and the standard scalar product $\langle \cdot, \cdot \rangle$ of $\mathbb{R}^m$. The *orthogonal lattice* of a lattice $\mathcal{L} \subseteq \mathbb{Z}^m$ is

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

We define the *completion* of a lattice $\mathcal{L}$ as the lattice $\bar{\mathcal{L}} = E_{\mathcal{L}} \cap \mathbb{Z}^m = (\mathcal{L}^\perp)^\perp$. Clearly, $\mathcal{L}$ is a full rank sublattice of $\bar{\mathcal{L}}$. We say that a lattice is *complete* if it coincides with its completion, i.e. $\bar{\mathcal{L}} = \mathcal{L}$. One can prove that $\dim \mathcal{L} + \dim \mathcal{L}^\perp = m$ and $\det(\mathcal{L}^\perp) = \det(\bar{\mathcal{L}}) \leq \det(\mathcal{L})$; we recall the proofs in the full version of this paper [CG20]. By Hadamard's inequality, we have $\det(\mathcal{L}) \leq \prod_{i=1}^{d} \|\mathbf{b}_i\|$ for any basis $\mathbf{b}_1, \ldots, \mathbf{b}_d$ of a lattice $\mathcal{L}$; this implies that $\det(\mathcal{L}^\perp) \leq \prod_{i=1}^{d} \|\mathbf{b}_i\|$ for any basis $\mathbf{b}_1, \ldots, \mathbf{b}_d$ of $\mathcal{L}$.

Lattice minima. The *first minimum* $\lambda_1(\mathcal{L})$ of a lattice $\mathcal{L}$ is the minimum of the norm of its non-zero vectors. Lattice points whose norm is $\lambda_1(\mathcal{L})$ are called *shortest vectors*. The *Hermite constant* γ_d, in dimension d, is the supremum of

$\lambda_1(\mathcal{L})^2/\det(\mathcal{L})^{\frac{2}{d}}$ over all the lattices of rank d. Using Minkowski convex body theorem, one can prove that for each $d \in \mathbb{N}^+$, $0 \leq \gamma_d \leq d/4 + 1$.

More generally, 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. *Minkowski's Second Theorem* states that for each $1 \leq i \leq d$

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

Lattice reduction. LLL-reduced bases have many good properties. In particular the first vector $\mathbf{b}_1$ of an LLL-reduced basis is not much longer than the shortest vector of the lattice.

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

The LLL algorithm [LLL82] outputs an LLL-reduced basis of a rank-d lattice in $\mathbb{Z}^m$ in time $\mathcal{O}(d^5 m \log^3 B)$, from a basis of vectors of norm less than B. This was further improved by Nguyen and Stehlé in [NS09] with a variant based on proven floating point arithmetic, called L^2, with complexity $\mathcal{O}(d^4 m(d + \log B) \log B)$ without fast arithmetic. In this paper, when we apply LLL, we always mean the L^2 variant. We denote by log the logarithm in base 2.

Heuristics. For a "random lattice" we expect $\lambda_1(\mathcal{L}) \approx \sqrt{d} \det(\mathcal{L})^{\frac{1}{d}}$ by the *Gaussian Heuristic* and all lattice minima to be approximately the same. Omitting the $\sqrt{d}$ factor, for a lattice $\mathcal{L}$ generated by a set of d "random" vectors in $\mathbb{Z}^m$ for $d < m$, we expect the lattice $\mathcal{L}$ to be of rank d, and the short vectors of $\mathcal{L}^{\perp}$ to have norm approximately $(\det \mathcal{L}^{\perp})^{1/(m-d)} \simeq (\det \mathcal{L})^{1/(m-d)} \simeq (\prod_{i=1}^{d} \|\mathbf{b}_i\|)^{1/(m-d)}$.

3   The Nguyen-Stern Algorithm

In this section we recall the Nguyen-Stern algorithm for solving the hidden subset sum problem. We explain why the algorithm has complexity exponential in n and provide the result of practical experiments. Then in Sect. 4 we will describe our polynomial-time algorithm.

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

$$\mathbf{h} = \alpha_1 \mathbf{x}_1 + \alpha_2 \mathbf{x}_2 + \cdots + \alpha_n \mathbf{x}_n \pmod{M} \tag{2}$$

we must recover the vector $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n) \in \mathbb{Z}_M^n$ and the vectors $\mathbf{x}_i \in \{0, 1\}^m$. The Nguyen-Stern algorithm proceeds in 2 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. From $\mathbf{h}$, the $\mathbf{x}_i$'s and M, recover the weights α_i.

3.1   First Step: Orthogonal Lattice Attack

The orthogonal lattice attack. The goal of the orthogonal lattice attack is to recover the hidden lattice $\bar{\mathcal{L}}_\mathbf{x}$, where $\mathcal{L}_\mathbf{x} \subset \mathbb{Z}^m$ is the lattice generated by the n vectors $\mathbf{x}_i$. Let $\mathcal{L}_0$ be the lattice of vectors orthogonal to $\mathbf{h}$ modulo M:

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

Following [NS99], the main observation is that if $\langle \mathbf{u}, \mathbf{h} \rangle \equiv 0 \pmod{M}$, then from (2) we obtain:

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

and therefore the vector $\mathbf{p_u} = (\langle \mathbf{u}, \mathbf{x}_1 \rangle, \ldots, \langle \mathbf{u}, \mathbf{x}_n \rangle)$ is orthogonal to the vector $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$ modulo M. 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 M, then we must have $\mathbf{p_u} = 0$ and therefore $\mathbf{u} \in \mathcal{L}_\mathbf{x}^\perp$.

   Therefore, the orthogonal lattice attack consists in first computing an LLL-reduced basis of the lattice $\mathcal{L}_0$. The first $m - n$ short vectors $\mathbf{u}_1, \ldots, \mathbf{u}_{m-n}$ will give us a generating set of the lattice $\mathcal{L}_\mathbf{x}^\perp$. Then one can compute a basis of the lattice $\bar{\mathcal{L}}_\mathbf{x} = (\mathcal{L}_\mathbf{x}^\perp)^\perp$. This gives the following algorithm, which is the first step of the Nguyen-Stern algorithm; we explain the main steps in more details below.

Algorithm 1. Orthogonal lattice attack [NS99]

Input: $\mathbf{h}, M, n, m$.
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** $(\mathbf{c}_1, \ldots, \mathbf{c}_n)$

Constructing a basis of $\mathcal{L}_0$. We first explain how to construct a basis of $\mathcal{L}_0$. If the modulus M is prime we can assume $h_1 \neq 0$, up to permutation of the coordinates; indeed the case $\mathbf{h} = \mathbf{0}$ is trivial. More generally, we assume $\gcd(h_1, M) = 1$. We write $\mathbf{u} = [u_1, \mathbf{u}']$ where $\mathbf{u}' \in \mathbb{Z}^{m-1}$. Similarly we write $\mathbf{h} = [h_1, \mathbf{h}']$ where $\mathbf{h}' \in \mathbb{Z}^{m-1}$. Since h_1 is invertible modulo M, we get:

$$\mathbf{u} \in \mathcal{L}_0 \iff u_1 \cdot h_1 + \langle \mathbf{u}', \mathbf{h}' \rangle \equiv 0 \pmod{M}$$
$$\iff u_1 + \langle \mathbf{u}', \mathbf{h}' \rangle \cdot h_1^{-1} \equiv 0 \pmod{M}$$

Therefore, a basis of $\mathcal{L}_0$ is given by the $m \times m$ matrix of row vectors:

$$\mathcal{L}_0 = \begin{bmatrix} M & \\ -\mathbf{h}' \cdot h_1^{-1}[M] & \mathbf{I}_{m-1} \end{bmatrix}$$

To compute a reduced basis $\mathbf{u}_1, \ldots, \mathbf{u}_m$ of the lattice $\mathcal{L}_0$ we use the L^2 algorithm. The complexity is then $\mathcal{O}(m^5(m + \log M) \log M)$ without fast arithmetic. We show in Sect. 3.2 below that for a sufficiently large modulus M, the first $m - n$ vectors $\mathbf{u}_1, \ldots, \mathbf{u}_{m-n}$ must form a generating set of $\mathcal{L}_{\mathbf{x}}^{\perp}$.

Computing a basis of $\bar{\mathcal{L}}_{\mathbf{x}} = (\mathcal{L}_{\mathbf{x}}^{\perp})^{\perp}$. From the vectors $\mathbf{u}_1, \ldots, \mathbf{u}_{m-n}$ forming a generating set of the lattice $\mathcal{L}_{\mathbf{x}}^{\perp}$, we can compute its orthogonal $\bar{\mathcal{L}}_{\mathbf{x}} = (\mathcal{L}_{\mathbf{x}}^{\perp})^{\perp}$ using the LLL-based algorithm from [NS97]. Given a lattice $\mathcal{L}$, the algorithm from [NS97] produces an LLL-reduced basis of $\mathcal{L}^{\perp}$ in polynomial time; we refer to the full version of this paper [CG20] for a detailed description of the algorithm. Therefore we obtain an LLL-reduced basis of $\bar{\mathcal{L}}_{\mathbf{x}} = (\mathcal{L}_{\mathbf{x}}^{\perp})^{\perp}$ in polynomial-time.

3.2   Rigorous Analysis of Step 1

We now provide a rigorous analysis of the orthogonal lattice attack above. More precisely, we show that for a large enough modulus M, the orthogonal lattice attack recovers a basis of $\bar{\mathcal{L}}_{\mathbf{x}}$ in polynomial time, for a significant fraction of the weight α_i's.

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 M is a prime integer of bitsize at least $2mn \log m$. For $m = 2n$, the density is $d = n / \log M = \mathcal{O}(1/(n \log n))$.*

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

Lemma 2. *Assume that the lattice $\mathcal{L}_{\mathbf{x}}$ has rank n. Algorithm 1 computes a basis of the lattice $\bar{\mathcal{L}}_{\mathbf{x}}$ in polynomial time under the condition $m > n$ and*

$$\sqrt{mn} \cdot 2^{\frac{m}{2}} \cdot \lambda_{m-n}\left(\mathcal{L}_{\mathbf{x}}^{\perp}\right) < \lambda_1\left(\Lambda_M^{\perp}(\boldsymbol{\alpha})\right). \tag{3}$$

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

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

is orthogonal to the vector $\boldsymbol{\alpha}$ modulo M; therefore if $\mathbf{p_u}$ is shorter than the shortest non-zero vector orthogonal to $\boldsymbol{\alpha}$ modulo M, we must have $\mathbf{p_u} = 0$, and therefore $\mathbf{u} \in \mathcal{L}_{\mathbf{x}}^{\perp}$; this happens under the condition $\|\mathbf{p_u}\| < \lambda_1\left(\Lambda_M^{\perp}(\boldsymbol{\alpha})\right)$. Since $\|\mathbf{p_u}\| \leq \sqrt{mn}\|\mathbf{u}\|$, given any $\mathbf{u} \in \mathcal{L}_0$ we must have $\mathbf{u} \in \mathcal{L}_{\mathbf{x}}^{\perp}$ under the condition:

$$\sqrt{mn}\|\mathbf{u}\| < \lambda_1\left(\Lambda_M^{\perp}(\boldsymbol{\alpha})\right). \tag{4}$$

The lattice $\mathcal{L}_0$ is full rank of dimension m since it contains $M\mathbb{Z}^m$. Now, 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

$$\|\mathbf{u}_j\| \leq 2^{\frac{m}{2}} \cdot \lambda_{m-n}(\mathcal{L}_0) \leq 2^{\frac{m}{2}} \cdot \lambda_{m-n}\left(\mathcal{L}_{\mathbf{x}}^{\perp}\right) \tag{5}$$

since $\mathcal{L}_{\mathbf{x}}^{\perp}$ is a sublattice of $\mathcal{L}_0$ of dimension $m - n$. Combining with (4), this implies that when

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

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 1 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. □

The following Lemma is based on a counting argument; we provide the proof in the full version of this paper [CG20].

Lemma 3. *Let M be a prime. Then with probability at least $1/2$ over the choice of $\boldsymbol{\alpha}$, we have $\lambda_1(\Lambda_M^{\perp}(\boldsymbol{\alpha})) \geq M^{1/n}/4$.* □

Combining the two previous lemmas, we can prove Theorem 1.

Proof (of Theorem 1). In order to apply Lemma 2, 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 $m - n$ and by Minkowski's second theorem we have

$$\lambda_{m-n}\left(\mathcal{L}_{\mathbf{x}}^{\perp}\right) \leq \sqrt{\gamma_{m-n}}^{m-n} \det\left(\mathcal{L}_{\mathbf{x}}^{\perp}\right) \leq m^{m/2} \det\left(\mathcal{L}_{\mathbf{x}}^{\perp}\right). \tag{6}$$

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

$$\det \mathcal{L}_{\mathbf{x}}^{\perp} \leq \det \mathcal{L}_{\mathbf{x}} \leq \prod_{i=1}^{n} \|\mathbf{x}_i\| \leq m^{n/2} \tag{7}$$

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

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

Thus, by Lemma 2, we can recover a basis of $\bar{\mathcal{L}}_{\mathbf{x}}$ when

$$\sqrt{mn} \cdot 2^{\frac{m}{2}} \cdot m^m < \lambda_1\left(\Lambda_M^{\perp}(\boldsymbol{\alpha})\right).$$

From Lemma 3, with probability at least $1/2$ over the choice of $\boldsymbol{\alpha}$ we can therefore recover the hidden lattice $\bar{\mathcal{L}}_{\mathbf{x}}$ if:

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

For $m > n \geq 4$, it suffices to have $\log M \geq 2mn \log m$. □

3.3   Heuristic Analysis of Step 1

In the previous section, we have shown that the orthogonal lattice attack provably recovers the hidden lattice $\bar{\mathcal{L}}_{\mathbf{x}}$ in polynomial time for a large enough modulus M, namely we can take $\log M = \mathcal{O}(n^2 \log n)$ when $m = 2n$. Below we show that heuristically we can take $\log M = \mathcal{O}(n^2)$, which gives a knapsack density $d = n/\log M = \mathcal{O}(1/n)$. We also give the concrete bitsize of M used in our experiments, and provide a heuristic complexity analysis.

Heuristic size of the modulus M. In order to derive a heuristic size for the modulus M, we use an approximation of the terms in the condition (3) from Lemma 2.

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 $\lambda_1\left(\mathcal{L}_{\mathbf{x}}^{\perp}\right)$. This means that instead of the rigorous inequality (6) from the proof of Theorem 1, we use the heuristic approximation:

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

Using (7), this gives:

$$\lambda_{m-n}\left(\mathcal{L}_{\mathbf{x}}^{\perp}\right) \lessgtr \sqrt{\gamma_{m-n}} m^{\frac{n}{2(m-n)}}. \tag{8}$$

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

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

Finally the $2^{m/2}$ factor in (3) 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 (3) we obtain the heuristic condition:

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

This gives the condition:

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

which gives:

$$\log M > \iota \cdot m \cdot n + \frac{n}{2} \log(n \cdot \gamma_{m-n}/\gamma_n) + \frac{mn}{2(m-n)} \log m. \tag{9}$$

If we consider $m = n + k$ for some constant k, we can take $\log M = \mathcal{O}(n^2 \log n)$. If $m > c \cdot n$ for some constant $c > 1$, we can take $\log M = \mathcal{O}(m \cdot n)$. In particular, for $m = 2n$ we obtain the condition:

$$\log M > 2\iota \cdot n^2 + \frac{3n}{2} \log n + n \tag{10}$$

which gives $\log M = \mathcal{O}(n^2)$ and a knapsack density $d = n/\log M = \mathcal{O}(1/n)$. In practice for our experiments we use $m = 2n$ and $\log M \simeq 2\iota n^2 + n \log n$ with $\iota = 0.035$. Finally, we note that smaller values of M could be achieved by using BKZ reduction of $\mathcal{L}_0$ instead of LLL.

Heuristic complexity. Recall that for a rank-d lattice in $\mathbb{Z}^m$, the complexity of computing an LLL-reduced basis with the L^2 algorithm is $\mathcal{O}(d^4 m(d + \log B) \log B)$ without fast integer arithmetic, for vectors of Euclidean norm less than B. At Step 1 we must apply LLL-reduction twice.

The first LLL is applied to the rank-m lattice $\mathcal{L}_0 \in \mathbb{Z}^m$. Therefore the complexity of the first LLL is $\mathcal{O}(m^5(m + \log M) \log M)$. If $m = n + k$ for some constant k, the heuristic complexity is therefore $\mathcal{O}(n^9 \log^2 n)$. If $m > c \cdot n$ for some constant c, the heuristic complexity is $\mathcal{O}(m^7 \cdot n^2)$.

The second LLL is applied to compute the orthogonal of $\mathcal{L}(\mathbf{U})$ where $\mathbf{U}$ is the matrix basis of the vectors $\mathbf{u}_1, \ldots, \mathbf{u}_{m-n} \in \mathbb{Z}^m$. From (5) and (8), we can heuristically assume $\|\mathbf{U}\| \le 2^{m/2} \cdot \sqrt{m} \cdot m^{\frac{n}{2(m-n)}}$. For $m = n + k$ for some constant k, this gives $\log \|\mathbf{U}\| = \mathcal{O}(n \log n)$, while for $m > c \cdot n$ for some constant $c > 1$, we obtain $\log \|\mathbf{U}\| = \mathcal{O}(m)$. The heuristic complexity of computing the orthogonal of $\mathbf{U}$ is $\mathcal{O}(m^5(m + (m/n) \log \|\mathbf{U}\|)^2)$ (see the full version of this paper [CG20]). For $m = n + k$, the complexity is therefore $\mathcal{O}(n^7 \log^2 n)$, while for $m > c \cdot n$, the complexity is $\mathcal{O}(m^9/n^2)$.

We summarize the complexities of the two LLL operations in Table 1; we see that the complexities are optimal for $m = c \cdot n$ for some constant $c > 1$, so for simplicity we take $m = 2n$. In that case the heuristic complexity of the first step is $\mathcal{O}(n^9)$, and the density is $d = n/\log M = \mathcal{O}(1/n)$, as in the classical subset-sum problem.

Table 1. Modulus size and time complexity of Algorithm 1 as a function of the parameter m.

m	$\log M$	LLL $\mathcal{L}_0$	LLL $(\mathcal{L}_{\mathbf{x}}^\perp)^\perp$
$\gg n$	$\mathcal{O}(n \cdot m)$	$\mathcal{O}(m^7 \cdot n^2)$	$\mathcal{O}(m^9/n^2)$
n^2	$\mathcal{O}(n^3)$	$\mathcal{O}(n^{16})$	$\mathcal{O}(n^{16})$
$2n$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^9)$	$\mathcal{O}(n^7)$
$n + 1$	$\mathcal{O}(n^2 \log n)$	$\mathcal{O}(n^9 \log^2 n)$	$\mathcal{O}(n^7 \log^2 n)$

3.4   Second Step of the Nguyen-Stern Attack

From the first step we have obtained 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 this does not necessarily reveal the vectors $\mathbf{x}_i$. Namely, because of 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, to recover the original vectors $\mathbf{x}_i$, one must apply BKZ instead of LLL, in order to obtain a better approximation factor; eventually from $\mathbf{h}$, the $\mathbf{x}_i$'s and M, one can recover the weights α_i by solving a linear system; this is the second step of the Nguyen-Stern algorithm.

The authors of [NS99] did not provide a time complexity analysis of their algorithm. In the following, we provide a heuristic analysis of the second step of the Nguyen-Stern algorithm, based on a model of the gap between the shortest vectors of $\mathcal{L}_{\mathbf{x}}$ (the vectors $\mathbf{x}_i$), and the "generic" short vectors of $\mathcal{L}_{\mathbf{x}}$. While relatively simplistic, our model seems to accurately predict the minimal block-size β required for BKZ reduction; we provide the result of practical experiments

in the next section. Under this model the BKZ block-size β must increase almost linearly with n; the complexity of the attack is therefore exponential in n. In our analysis below, for simplicity we heuristically assume that the lattice $\mathcal{L}_{\mathbf{x}}$ is complete, i.e. $\bar{\mathcal{L}}_{\mathbf{x}} = \mathcal{L}_{\mathbf{x}}$.

Short vectors in $\mathcal{L}_{\mathbf{x}}$. The average norm of the original binary vectors $\mathbf{x}_i \in \mathbb{Z}^m$ is roughly $\sqrt{m/2}$. If we take the difference between some $\mathbf{x}_i$ and $\mathbf{x}_j$, the components remain in $\{-1, 0, 1\}$, and the average norm is also roughly $\sqrt{m/2}$. Therefore, we can assume that the vectors $\mathbf{x}_i$ and $\mathbf{x}_i - \mathbf{x}_j$ for $i \neq j$ are the shortest vectors of the lattice $\mathcal{L}_{\mathbf{x}}$.

We can construct "generic" short vectors in $\mathcal{L}_{\mathbf{x}}$ by taking a linear combination with $\{0, 1\}$ coefficients of vectors of the form $\mathbf{x}_i - \mathbf{x}_j$. For $\mathbf{x}_i - \mathbf{x}_j$, the variance of each component is $1/2$. If we take a linear combination of $n/4$ such differences (so that roughly half of the coefficients with respect to the vectors $\mathbf{x}_i$ are 0), the variance for each component will be $n/4 \cdot 1/2 = n/8$, and for m components the norm of the resulting vector will be about $\sqrt{nm/8}$. Therefore heuristically the gap between these generic vectors and the shortest vectors is:

$$\frac{\sqrt{nm/8}}{\sqrt{m/2}} = \frac{\sqrt{n}}{2}.$$

Running time with BKZ. To recover the shortest vectors, the BKZ approximation factor $2^{\iota \cdot n}$ should be less than the above gap, which gives the condition:

$$2^{\iota \cdot n} \leq \frac{\sqrt{n}}{2} \tag{11}$$

which gives $\iota \leq (\log(n/4))/(2n)$. 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]. Therefore the running time of the Nguyen-Stern algorithm is $2^{\Omega(n/\log n)}$, with BKZ block-size $\beta = \omega(n/\log n)$ in the second step.

Recovering the vectors $\mathbf{x}_i$. It remains to show how to recover the vectors $\mathbf{x}_i$. Namely as explained above the binary vectors $\mathbf{x}_i$ are not the only short vectors in $\mathcal{L}_{\mathbf{x}}$; the vectors $\mathbf{x}_i - \mathbf{x}_j$ are roughly equally short. The approach from [NS99] is as follows. Since the short vectors in $\mathcal{L}_{\mathbf{x}}$ probably have components in $\{-1, 0, 1\}$, the authors suggest to transform the lattice $\mathcal{L}_{\mathbf{x}}$ into a new one $\mathcal{L}'_{\mathbf{x}} = 2\mathcal{L}_{\mathbf{x}} + \mathbf{e}\mathbb{Z}$, where $\mathbf{e} = (1, \ldots, 1)$. Namely in that case a vector $\mathbf{v} \in \mathcal{L}_{\mathbf{x}}$ with components in $\{-1, 0, 1\}$ will give a vector $2\mathbf{v} \in \mathcal{L}'_{\mathbf{x}}$ with components in $\{-2, 0, 2\}$, whereas a vector $\mathbf{x} \in \mathcal{L}_{\mathbf{x}}$ with components in $\{0, 1\}$ will give a vector $2\mathbf{x} - \mathbf{e} \in \mathcal{L}'_{\mathbf{x}}$ with components in $\{-1, 1\}$, hence shorter. This should enable to recover the secret vectors $\mathbf{x}_i$ as the shortest vectors in $\mathcal{L}'_{\mathbf{x}}$.

Below we describe a slightly simpler approach in which we stay in the lattice $\mathcal{L}_{\mathbf{x}}$. First, we explain why for large enough values of m, we are unlikely to obtain vectors in $\{0, \pm 1\}$ as combination of more that two $\mathbf{x}_i$'s. Namely if we take a

linear combination of the form $\mathbf{x}_i - \mathbf{x}_j + \mathbf{x}_k$, each component will be in $\{-1, 0, 1\}$ with probability 7/8; therefore for m components the probability will be $(7/8)^m$. There are at most n^3 such triples to consider, so we want $n^3 \cdot (7/8)^m < \varepsilon$, which gives the condition $m \geq 16 \log n - 6 \log \varepsilon$. With $m = 2n$ and $\varepsilon = 2^{-4}$, this condition is satisfied for $n \geq 60$; for smaller values of n, one should take $m = \max(2n, 16 \log n + 24)$.

Hence after BKZ reduction with a large enough block-size β as above, we expect that each of the basis vectors $(\mathbf{c}_1, \ldots, \mathbf{c}_n)$ is either equal to $\pm\mathbf{x}_i$, or equal to a combination of the form $\mathbf{x}_i - \mathbf{x}_j$ for $i \neq j$. Conversely, this implies that all rows of the transition matrix between $(\mathbf{c}_1, \ldots, \mathbf{c}_n)$ and $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ must have Hamming weight at most 4. Therefore while staying in the lattice $\mathcal{L}_{\mathbf{x}}$ we can recover each of the original binary vectors $\mathbf{x}_i$ from the basis vectors $(\mathbf{c}_1, \ldots, \mathbf{c}_n)$, by exhaustive search with $\mathcal{O}(n^4)$ tests. In the full version of this paper [CG20] we describe a greedy algorithm that recovers the original binary vectors $\mathbf{x}_i$ relatively efficiently.

Recovering the weights α_i. Finally, from the samples $\mathbf{h}$, the vectors $\mathbf{x}_i$'s and the modulus M, recovering the weights α_i is straightforward as this amounts to solving a linear system:

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

Letting $\mathbf{X}'$ be the $n \times n$ matrix with the first n components of the column vectors $\mathbf{x}_i$ and letting $\mathbf{h}'$ be the vector with the first n components of $\mathbf{h}$, we have $\mathbf{h}' = \mathbf{X}' \cdot \boldsymbol{\alpha}$ where $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n) \pmod{M}$. Assuming that $\mathbf{X}'$ is invertible modulo M, we get $\boldsymbol{\alpha} = \mathbf{X}'^{-1} \mathbf{h}' \pmod{M}$.

3.5   Practical Experiments

Running times. We provide in Table 2 the result of practical experiments. The first step is the orthogonal lattice attack with two applications of LLL. For the second step, we receive as input from Step 1 an LLL-reduced basis of the lattice $\mathcal{L}_{\mathbf{x}}$. We see in Table 2 that for $n = 70$ this is sufficient to recover the hidden vectors $\mathbf{x}_i$. Otherwise, we apply BKZ with block-size $\beta = 10, 20, 30, \ldots$ until we recover the vectors $\mathbf{x}_i$. We see that the two LLLs from Step 1 run in reasonable time up to $n = 250$, while for Step 2 the running time of BKZ grows exponentially, so we could not run Step 2 for $n > 170$. We provide the source code of our `SageMath` implementation in https://pastebin.com/ZFk1qjfP, based on the L^2 [NS09] and BKZ 2.0 [CN11] implementations from [fpl16].

Hermite factors. Recall that from our heuristic model from Sect. 3.4 the target Hermite factor for the second step of the Nguyen-Stern algorithm is $\gamma = \sqrt{n}/2$, which can be written $\gamma = a^n$ with $a = (n/4)^{1/(2n)}$. We provide in Table 2 above the corresponding target Hermite factors as a function of n.

In order to predict the Hermite factor achievable by BKZ as a function of the block-size β, we have run some experiments on a different lattice, independent

Table 2. Running time of the [NS99] attack, under a 3,2 GHz Intel Core i5 processor.

			Step 1		Step 2			
n	m	$\log M$	LLL $\mathcal{L}_0$	LLL $\mathcal{L}_{\mathbf{x}}^{\perp}$	Hermite	Reduction		Total
70	140	772	3 s	1 s	1.021^n	LLL	ε	6 s
90	180	1151	10 s	4 s	1.017^n	BKZ-10	1 s	18 s
110	220	1592	28 s	12 s	1.015^n	BKZ-10	3 s	50 s
130	260	2095	81 s	24 s	1.013^n	BKZ-20	10 s	127 s
150	300	2659	159 s	44 s	1.012^n	BKZ-30	4 min	8 min
170	340	3282	6 min	115 s	1.011^n	BKZ-30	438 min	447 min
190	380	3965	13 min	3 min	1.010^n	–	–	–
220	440	5099	63 min	29 min	1.009^n	–	–	–
250	500	6366	119 min	56 min	1.008^n	–	–	–

from our model of Sect. 3.4. For this we have considered the lattice $\mathcal{L} \in \mathbb{Z}^n$ of row vectors:

$$
\mathcal{L} = \begin{bmatrix}
p & & & & \\
c_1 & 1 & & & \\
c_2 & & 1 & & \\
\vdots & & & \ddots & \\
c_{n-1} & \cdots & & & 1
\end{bmatrix}
$$

for some prime p, with random c_i's modulo p. Since $\det \mathcal{L} = p$, by applying LLL or BKZ we expect to obtain vectors of norm $2^{\iota n}(\det L)^{1/n} = 2^{\iota n} \cdot p^{1/n}$, where $2^{\iota n}$ is the Hermite factor. We summarize our results in Table 3 below. Values up to $\beta = 40$ are from our experiments with the lattice $\mathcal{L}$ above, while for $\beta \geq 85$ the values are reproduced from [CN11], based on a simulation approach.

Table 3. Experimental and simulated Hermite factors for LLL ($\beta = 2$) and for BKZ with block-size β.

Block-size β	2	10	20	30	40	85	106	133
Hermite factor	1.020^n	1.015^n	1.014^n	1.013^n	1.012^n	1.010^n	1.009^n	1.008^n

In summary, the minimal BKZ block-sizes β required experimentally in Table 2 to apply Step 2 of Nguyen-Stern, seem coherent with the target Hermite factors from our model, and the experimental Hermite factors from Table 3. For example, for $n = 70$, this explains why an LLL-reduced basis is sufficient, because the target Hermite factor is 1.021^n, while LLL can achieve 1.020^n. From Table 3, BKZ-10 can achieve 1.015^n, so in Table 2 it was able to break the instances $n = 90$ and $n = 110$, but not $n = 130$ which has target Hermite factor 1.013^n. However we see that BKZ-20 and BKZ-30 worked better than expected; for example BKZ-30 could break the instance $n = 170$ with target Hermite factor 1.011^n, while in principle from Table 3 it can only achieve 1.013^n. So it could

be that our model from Sect. 3.4 underestimates the target Hermite factor. Nevertheless, we believe that our model and the above experiments confirm that the complexity of the Nguyen-Stern algorithm is indeed exponential in n.

4   Our Polynomial-Time Algorithm for Solving the Hidden Subset-Sum Problem

Recall that the Nguyen-Stern attack is divided in the two following 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. From $\mathbf{h}$, the $\mathbf{x}_i$'s and M, recover the weights α_i.

In the previous section we have argued that the complexity of the second step of the Nguyen-Stern attack is exponential in n. In this section we describe an alternative second step with polynomial-time complexity. However, our second step requires more samples than in [NS99], namely we need $m \simeq n^2/2$ samples instead of $m = 2n$. This means that in the first step we must produce a basis of the rank-n lattice $\bar{\mathcal{L}}_{\mathbf{x}} \subset \mathbb{Z}^m$, with the much higher vector dimension $m \simeq n^2/2$ instead of $m = 2n$.

For this, the naive method would be to apply directly Algorithm 1 from Sect. 3.1 to the vector $\mathbf{h}$ of dimension $m \simeq n^2/2$. But for $n \simeq 200$ one would need to apply LLL on a $m \times m$ matrix with $m \simeq n^2/2 \simeq 20\,000$, which is not practical; moreover the bitsize of the modulus M would need to be much larger due to the Hermite factor of LLL in such large dimension (see Table 1). Therefore, we first explain how to modify Step 1 in order to efficiently generate a lattice basis of $\bar{\mathcal{L}}_{\mathbf{x}} \subset \mathbb{Z}^m$ for large m. Our technique is as follows: instead of applying LLL on a square matrix of dimension $n^2/2$, we apply LLL in parallel on $n/2$ square matrices of dimension $2n$, which is much faster. Eventually we show in Sect. 5 that a single application of LLL is sufficient.

4.1   First Step: Obtaining a Basis of $\bar{\mathcal{L}}_{\mathbf{x}}$ for $m \gg n$

In this section, we show how to adapt the first step, namely the orthogonal lattice attack from [NS99] recalled in Sect. 3.1, to the case $m \gg n$. More precisely, we show how to generate a basis of n vectors of $\bar{\mathcal{L}}_{\mathbf{x}} \subset \mathbb{Z}^m$ for $m \simeq n^2/2$, while applying LLL on matrices of dimension $t = 2n$ only. As illustrated in Fig. 1, this is relatively straightforward: we apply Algorithm 1 from Sect. 3.1 on $2n$ components of the vector $\mathbf{h} \in \mathbb{Z}^m$ at a time, and each time we recover roughly the projection of a lattice basis of $\bar{\mathcal{L}}_{\mathbf{x}}$ on those $2n$ components; eventually we recombine those projections to obtain a full lattice basis of $\bar{\mathcal{L}}_{\mathbf{x}}$.

More precisely, writing $\mathbf{h} = [\mathbf{h}_0, \dots, \mathbf{h}_d]$ where $m = (d+1) \cdot n$ and $\mathbf{h}_i \in \mathbb{Z}^n$, we apply Algorithm 1 on each of the d sub-vectors of the form $(\mathbf{h}_0, \mathbf{h}_i) \in \mathbb{Z}^{2n}$ for $1 \leq i \leq d$. For each $1 \leq i \leq d$ this gives us $\mathbf{C}_0^{(i)} \| \mathbf{C}_i \in \mathbb{Z}^{n \times 2n}$, the completion
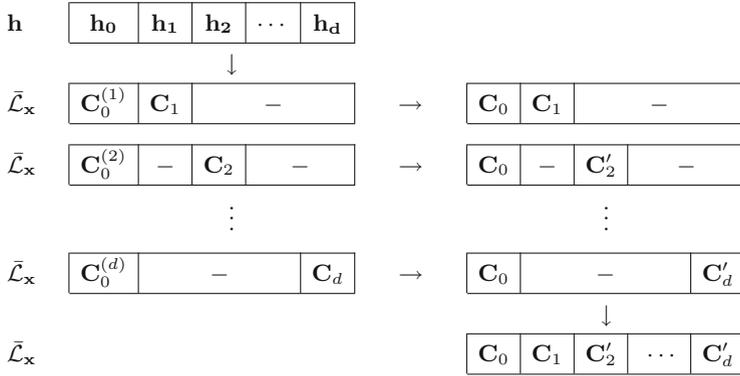
Fig. 1. Computation of a lattice basis of $\bar{\mathcal{L}}_{\mathbf{x}}$.

of the projection of a lattice basis of $\mathcal{L}_{\mathbf{x}}$. To recover the m components of the basis, we simply need to ensure that the projected bases $\mathbf{C}_0^{(i)}\|\mathbf{C}_i \in \mathbb{Z}^{n \times 2n}$ always start with the same matrix $\mathbf{C}_0$ on the first n components; see Fig. 1 for an illustration. This gives Algorithm 2 below. We denote Algorithm 1 from Sect. 3.1 by OrthoLat.

Algorithm 2. Orthogonal lattice attack with $m = d \cdot n$ samples

---

Input: $\mathbf{h} \in \mathbb{Z}^m, M, n, m = d \cdot n$.
Output: A base matrix of $\bar{\mathcal{L}}_{\mathbf{x}}$.
 1: Write $\mathbf{h} = [\mathbf{h}_0, \dots, \mathbf{h}_d]$ where $\mathbf{h}_i \in \mathbb{Z}^n$ for all $0 \leq i \leq d$.
 2: **for** $i \leftarrow 1$ to d **do**
 3:     $\mathbf{y}_i \leftarrow [\mathbf{h}_0, \mathbf{h}_i]$
 4:     $\mathbf{C}_0^{(i)}\|\mathbf{C}_i \leftarrow \mathsf{OrthoLat}(\mathbf{y}_i, M, n, 2n)$
 5:     $\mathbf{Q}_i \leftarrow \mathbf{C}_0^{(1)} \cdot (\mathbf{C}_0^{(i)})^{-1}$
 6:     $\mathbf{C}_i' \leftarrow \mathbf{Q}_i \cdot \mathbf{C}_i$
 7: **end for**
 8: **return** $[\mathbf{C}_0, \mathbf{C}_1, \mathbf{C}_2', \cdots, \mathbf{C}_d']$

A minor difficulty is that in principle, when applying OrthoLat (Algorithm 1) to a subset $\mathbf{y}_i \in \mathbb{Z}^{2n}$ of the sample $\mathbf{h} \in \mathbb{Z}^m$, we actually recover the completion of the projection of $\mathcal{L}_{\mathbf{x}}$ over the corresponding coordinates, rather than the projection of the completion $\bar{\mathcal{L}}_{\mathbf{x}}$ of $\mathcal{L}_{\mathbf{x}}$. More precisely, denote by π a generic projection on some coordinates of a lattice $\mathcal{L}_{\mathbf{x}}$. It is always true that $\pi(\mathcal{L}_{\mathbf{x}}) \subseteq \pi(\bar{\mathcal{L}}_{\mathbf{x}}) \subseteq \overline{\pi(\mathcal{L}_{\mathbf{x}})}$. Thus applying Algorithm 1 with a certain projection π we recover the completion $\overline{\pi(\mathcal{L}_{\mathbf{x}})}$. Assuming that the projection $\pi(\mathcal{L}_{\mathbf{x}})$ is complete, we obtain $\pi(\mathcal{L}_{\mathbf{x}}) = \overline{\pi(\mathcal{L}_{\mathbf{x}})} = \pi(\bar{\mathcal{L}}_{\mathbf{x}})$. Therefore, to simplify the analysis of Algorithm 2, we assume that the projection over the first n coordinates has rank n, and that the projection over the first $2n$ coordinates is complete. This implies that

the transition matrices $\mathbf{Q}_i \leftarrow \mathbf{C}_0^{(1)} \cdot (\mathbf{C}_0^{(i)})^{-1}$ for $2 \leq i \leq d$ must be integral; in our practical experiments this was always the case.

Theorem 2. *Let $m = d \cdot n$ for $d \in \mathbb{N}$ and $d > 1$. Assume that the projection of the lattice $\mathcal{L}_\mathbf{x} \in \mathbb{Z}^m$ over the first n components has rank n, and that the projection of $\mathcal{L}_\mathbf{x}$ over the first $2n$ coordinates is complete. With probability at least $1/2$ over the choice of $\boldsymbol{\alpha}$, Algorithm 2 recovers a basis of $\bar{\mathcal{L}}_\mathbf{x}$ in polynomial time, assuming that M is a prime of bitsize at least $4n^2(\log n + 1)$.*

Proof. From Theorem 1, we recover for each $1 \leq i \leq d$ a basis $\mathbf{C}_0^{(i)} \| \mathbf{C}_i$ corresponding to the completed projection of $\mathcal{L}_\mathbf{x}$ to the first n coordinates and the $i + 1$-th subset of n coordinates, with probability at least $1/2$ over the choice of $\boldsymbol{\alpha}$. Let us denote by $\mathbf{X}$ the base matrix whose rows are the vectors $\mathbf{x}_i$'s. By assumption the vectors $\mathbf{x}_i$ are linear independent, the first $n \times n$ minor $\mathbf{X}_0$ is invertible and the matrices $\mathbf{C}_0^{(i)}$ for $i = 1, \ldots, d$ must generate a superlattice of $\mathbf{X}_0$. In particular, there exists an invertible integral matrix $\mathbf{Q}_i$ such that $\mathbf{Q}_i \cdot \mathbf{C}_0^{(i)} = \mathbf{C}_0^{(1)}$ for each $i = 1, \ldots, d$. So, applying $\mathbf{Q}_i = \mathbf{C}_0^{(1)}(\mathbf{C}_0^{(i)})^{-1}$ to $\mathbf{C}_i$ we find $\mathbf{C}_i'$, which contains the $i + 1$-th subset of n coordinates of the vectors in a basis having $\mathbf{C}_0 := \mathbf{C}_0^{(1)}$ as projection on the first n coordinates. This implies that $[\mathbf{C}_0, \mathbf{C}_1, \mathbf{C}_2', \cdots \mathbf{C}_d']$ is a basis of $\bar{\mathcal{L}}_\mathbf{x}$. $\square$

Heuristic analysis. For the size of the modulus M, since we are working with lattices in $\mathbb{Z}^{2n}$, we can take the same modulus size as in the heuristic analysis of Step 1 from Sect. 3.3, namely

$$\log M \simeq 2\iota n^2 + n \log n$$

with $\iota = 0.035$. The time complexity of Algorithm 2 is dominated by the cost of applying OrthoLat (Algorithm 1) to each $\mathbf{y}_i$, which is heuristically $\mathcal{O}(n^9)$ from Sect. 3.3. Therefore, the heuristic complexity of Algorithm 2 is $d \cdot \mathcal{O}(n^9) = \mathcal{O}(m \cdot n^8)$. In particular, for $m \simeq n^2/2$, the heuristic complexity of Algorithm 2 is $\mathcal{O}(n^{10})$, instead of $\mathcal{O}(n^{16})$ with the naive method (see Table 1). In Sect. 5 we will describe an improved algorithm with complexity $\mathcal{O}(n^9)$.

4.2 Second Step: Recovering the Hidden Vectors $\mathbf{x}_i$'s

By the first step we recover a basis $\mathbf{C} = (\mathbf{c}_1, \ldots, \mathbf{c}_n)$ of the hidden lattice $\bar{\mathcal{L}}_\mathbf{x} \in \mathbb{Z}^m$. The goal of the second step is then to recover the original vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \bar{\mathcal{L}}_\mathbf{x}$, namely to solve the following problem:

Problem 1. *Let $\mathbf{X} \leftarrow \{0,1\}^{n \times m}$. Given $\mathbf{C} \in \mathbb{Z}^{n \times m}$ such that $\mathbf{WC} = \mathbf{X}$ for some $\mathbf{W} \in \mathbb{Z}^{n \times n} \cap \mathrm{GL}(\mathbb{Q}, n)$, recover $\mathbf{W}$ and $\mathbf{X}$.*

We show that for $m \simeq n^2/2$ the above problem can be solved in heuristic polynomial time, using a multivariate approach. Namely we reduce the problem to solving a system of multivariate quadratic equations and we provide an appropriate algorithm to solve it.

Heuristically we expect the solution to be unique up to permutations of the rows when $m \gg n$. Indeed for large enough m we expect the vectors $\mathbf{x}_i$ to be the unique vectors in $\bar{\mathcal{L}}_{\mathbf{x}}$ with binary coefficients. More precisely, consider a vector $\mathbf{v} = \mathbf{x}_i + \mathbf{x}_j$ or $\mathbf{v} = \mathbf{x}_i - \mathbf{x}_j$ for $i \neq j$. The probability that all components of $\mathbf{v}$ are in $\{0, 1\}$ is $(3/4)^m$, so for $n^2/2$ possible choices of i, j the probability is at most $n^2 \cdot (3/4)^m$, which for $m \simeq n^2/2$ is a negligible function of n. Therefore we can consider the equivalent problem:

Problem 2. *Given* $\mathbf{C} \in \mathbb{Z}^{n \times m}$ *of rank* n, *suppose there exist exactly* n *vectors* $\mathbf{w}_i \in \mathbb{Z}^n$ *such that* $\mathbf{w}_i \cdot \mathbf{C} = \mathbf{x}_i \in \{0, 1\}^m$ *for* $i = 1, \ldots, n$, *and assume that the vectors* $\mathbf{w}_i$ *are linearly independent. Find* $\mathbf{w}_1, \ldots, \mathbf{w}_n$.

We denote by $\tilde{\mathbf{c}}_1, \ldots, \tilde{\mathbf{c}}_m$ the column vectors of $\mathbf{C}$, which gives:

$$
\begin{bmatrix} \mathbf{w}_1 \\ \vdots \\ \mathbf{w}_n \end{bmatrix}
\begin{bmatrix} \tilde{\mathbf{c}}_1 & \cdots & \tilde{\mathbf{c}}_m \end{bmatrix}
=
\begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_n \end{bmatrix}
$$

Multivariate approach. The crucial observation is that since all components of the vectors $\mathbf{x}_i$ are binary, they must all satisfy the quadratic equation $y^2 - y = 0$. Therefore for each $i = 1, \ldots, n$ we have:

$$
\begin{aligned}
\mathbf{w}_i \cdot \mathbf{C} \in \{0, 1\}^m &\iff \forall j \in [1, m], \ (\mathbf{w}_i \cdot \tilde{\mathbf{c}}_j)^2 - \mathbf{w}_i \cdot \tilde{\mathbf{c}}_j = 0 \\
&\iff \forall j \in [1, m], \ (\mathbf{w}_i \cdot \tilde{\mathbf{c}}_j)(\mathbf{w}_i \cdot \tilde{\mathbf{c}}_j)^\mathsf{T} - \mathbf{w}_i \cdot \tilde{\mathbf{c}}_j = 0 \\
&\iff \forall j \in [1, m], \ \mathbf{w}_i \cdot (\tilde{\mathbf{c}}_j \cdot \tilde{\mathbf{c}}_j^\mathsf{T}) \cdot \mathbf{w}_i^\mathsf{T} - \mathbf{w}_i \cdot \tilde{\mathbf{c}}_j = 0
\end{aligned}
$$

Given the known column vectors $\tilde{\mathbf{c}}_1, \ldots, \tilde{\mathbf{c}}_m$, the vectors $\mathbf{w}_1, \ldots, \mathbf{w}_n$ and $\mathbf{0}$ are therefore solutions of the quadratic polynomial multivariate system

$$
\begin{cases}
\mathbf{w} \cdot \tilde{\mathbf{c}}_1 \tilde{\mathbf{c}}_1^\mathsf{T} \cdot \mathbf{w}^\mathsf{T} - \mathbf{w} \cdot \tilde{\mathbf{c}}_1 = 0 \\
\qquad \vdots \\
\mathbf{w} \cdot \tilde{\mathbf{c}}_m \tilde{\mathbf{c}}_m^\mathsf{T} \cdot \mathbf{w}^\mathsf{T} - \mathbf{w} \cdot \tilde{\mathbf{c}}_m = 0
\end{cases}
\tag{12}
$$

In the following we provide a heuristic polynomial-time algorithm to solve this quadratic multivariate system, via linearization and computation of eigenspaces. More precisely, as in the XL algorithm [CKPS00] we first linearize (12); then we prove that the $\mathbf{w}_i$'s are eigenvectors of some submatrices of the kernel matrix, and we provide a method to recover them in polynomial time. We observe that such approach is deeply related to Gröbner basis techniques for zero dimensional ideals. Namely, the system (12) of polynomial equations defines an ideal J. If the homogeneous degree 2 parts of such polynomials generate the space of monomials of degree 2, a Gröbner basis of J can be obtained via linear transformations, and the $\mathbf{x}_i$'s recovered in polynomial time. We refer to [CLO05] for the Gröbner basis perspective. For this approach the minimal condition is clearly $m = (n^2 + n)/2$.

Linearization. Since $(\tilde{\mathbf{c}}_j)_i = \mathbf{C}_{ij}$, for all $1 \le j \le m$, we can write:

$$\mathbf{y} \cdot \tilde{\mathbf{c}}_j \tilde{\mathbf{c}}_j^\mathsf{T} \cdot \mathbf{y}^\mathsf{T} = \sum_{i=1}^n \sum_{k=1}^n y_i y_k \mathbf{C}_{ij} \mathbf{C}_{kj} = \sum_{i=1}^n \sum_{k=i}^n y_i y_k (2 - \delta_{i,k}) \mathbf{C}_{ij} \mathbf{C}_{kj}$$

with $\delta_{i,k} = 1$ if $i = k$ and 0 otherwise. In the above equation the coefficient of the degree 2 monomial $y_i y_k$ for $1 \le i \le k \le n$ is $(2 - \delta_{i,k}) \mathbf{C}_{ij} \mathbf{C}_{kj}$. Thus, we consider the corresponding vectors of coefficients for $1 \le j \le m$:

$$\mathbf{r}_j = ((2 - \delta_{i,k}) \mathbf{C}_{ij} \mathbf{C}_{kj})_{1 \le i \le k \le n} \in \mathbb{Z}^{\frac{n^2+n}{2}}. \tag{13}$$

We set $\mathbf{R} \in \mathbb{Z}^{\frac{n^2+n}{2} \times m}$ to be the matrix whose columns are the $\mathbf{r}_j$'s and

$$\mathbf{E} = \left[\begin{array}{c} \mathbf{R} \\ \hline -\mathbf{C} \end{array} \right] \in \mathbb{Z}^{\frac{n^2+3n}{2} \times m};$$

we obtain that (12) is equivalent to

$$\begin{cases} [\; \mathbf{z} \mid \mathbf{y} \;] \cdot \mathbf{E} = 0 \\ \mathbf{z} = (y_i y_k)_{1 \le i \le k \le n} \in \mathbb{Z}^{\frac{n^2+n}{2}} \end{cases} \tag{14}$$

For $m > (n^2 + n)/2$ we expect the matrix $\mathbf{R}$ to be of rank $(n^2+n)/2$. In that case we must have $\operatorname{rank} \mathbf{E} \ge (n^2 + n)/2$, and so $\dim \ker \mathbf{E} \le n$. On the other hand, consider the set of vectors

$$\mathcal{W} = \{((w_i w_k)_{1 \le i \le k \le n}, \mathbf{w}) \in \mathbb{Z}^{\frac{n^2+3n}{2}} \mid \mathbf{w} \in \{\mathbf{w}_1, \dots, \mathbf{w}_n\}\}.$$

Since by assumption the vectors $\mathbf{w}_i$'s are linearly independent, $\operatorname{Span}(\mathcal{W})$ is a subspace of dimension n of $\ker \mathbf{E}$. This implies that $\dim \ker \mathbf{E} = n$, and that a basis of $\ker \mathbf{E}$ is given by the set $\mathcal{W}$. In the following, we show how to recover $\mathcal{W}$, from which we recover the matrix $\mathbf{W}$ and eventually the n vectors $\mathbf{x}_i$.

Kernel computation. Since the set of n vectors in $\mathcal{W}$ form a basis of $\ker \mathbf{E}$, the first step is to compute a basis of $\ker \mathbf{E}$ over $\mathbb{Q}$ from the known matrix $\mathbf{E} \in \mathbb{Z}^{\frac{n^2+3n}{2} \times m}$. However this does not immediately reveal $\mathcal{W}$ since the n vectors of $\mathcal{W}$ form a privileged basis of $\ker \mathbf{E}$; namely the vectors in $\mathcal{W}$ have the following structure:

$$\left((w_i w_k)_{1 \le i \le k \le n}, w_1, \dots w_n\right) \in \mathbb{Z}^{\frac{n^2+3n}{2}}.$$

To recover the vectors in $\mathcal{W}$ we proceed as follows. Note that the last n components in the vectors in $\mathcal{W}$ correspond to the linear part in the quadratic equations of (12). Therefore we consider the base matrix $\mathbf{K} \in \mathbb{Q}^{n \times \frac{n^2+3n}{2}}$ of $\ker \mathbf{E}$ such that the matrix corresponding to the linear part is the identity matrix:

$$\mathbf{K} = \left[\; \mathbf{M} \mid \mathbf{I}_n \right] \tag{15}$$

where $\mathbf{M} \in \mathbb{Q}^{n \times \frac{n^2+n}{2}}$. A vector $\mathbf{v} = (v_1, \ldots, v_n) \in \mathbb{Z}^n$ is then a solution of (14) if and only if $\mathbf{v} \cdot \mathbf{K} \in \mathcal{W}$, which gives:

$$\mathbf{v} \cdot \mathbf{M} = (v_i v_k)_{1 \le i \le k \le n}.$$

By duplicating some columns of the matrix $\mathbf{M}$, we can obtain a matrix $\mathbf{M}' \in \mathbb{Z}^{n^2 \times n}$ such that:

$$\mathbf{v} \cdot \mathbf{M}' = (v_i v_k)_{1 \le i \le n, 1 \le k \le n}.$$

We write $\mathbf{M}' = [\mathbf{M}_1, \ldots, \mathbf{M}_n]$ where $\mathbf{M}_i \in \mathbb{Z}^{n \times n}$. This gives:

$$\mathbf{v} \cdot \mathbf{M}_i = v_i \cdot \mathbf{v}$$

for all $1 \le i \le n$.

This means that the eigenvalues of each $\mathbf{M}_i$ are exactly all the possible i-th coordinates of the target vectors $\mathbf{w}_1, \ldots, \mathbf{w}_n$. Therefore the vectors $\mathbf{w}_j$'s are the intersections of the left eigenspaces corresponding to their coordinates.

Eigenspace computation. Consider for example the first coordinates $w_{j,1}$ of the vectors $\mathbf{w}_j$. From the previous equation, we have:

$$\mathbf{w}_j \cdot \mathbf{M}_1 = w_{j,1} \cdot \mathbf{w}_j.$$

Therefore the vectors $\mathbf{w}_j$ are the eigenvectors of the matrix $\mathbf{M}_1$, and their first coordinates $w_{j,1}$ are the eigenvalues. Assume that those n eigenvalues are distinct; in that case we can immediately compute the n corresponding eigenvectors $\mathbf{w}_j$ and solve the problem. More generally, we can recover the vectors $\mathbf{w}_j$ that belong to a dimension 1 eigenspace of $\mathbf{M}_1$; namely in that case $\mathbf{w}_j$ is the unique vector of its eigenspace such that $\mathbf{w}_j \cdot \mathbf{C} \in \{0,1\}^m$, and we recover the corresponding $\mathbf{x}_j = \mathbf{w}_j \cdot \mathbf{C}$.

Our approach is therefore as follows. We first compute the eigenspaces $E_1, \ldots, E_s$ of $\mathbf{M}_1$. For every $1 \le k \le s$, if $\dim E_k = 1$ then we can compute the corresponding target vector, as explained above. Otherwise, we compute $\mathbf{M}_{2,k}$ the restriction map of $\mathbf{M}_2$ at E_k and we check the dimensions of its eigenspaces. As we find eigenspaces of dimension 1 we compute more target vectors, otherwise we compute the restrictions of $\mathbf{M}_3$ at the new eigenspaces and so on. We iterate this process until we find all the solutions; see Algorithm 3 below.

In order to better analyze this procedure, we observe that we essentially construct a tree of subspaces of $\mathbb{Q}^n$, performing a breadth-first search algorithm. The root corresponds to the entire space, and each node at depth i is a son of a node E at depth $i - 1$ if and only if it represents a non-trivial intersection of E with one of the eigenspaces of $\mathbf{M}_i$. Since these non-trivial intersections are exactly the eigenspaces of the restriction of $\mathbf{M}_i$ to E, our algorithm does not compute unnecessary intersections. Moreover, we know that when the dimension of the node is 1 all its successors represent the same space; hence that branch of the algorithm can be closed; see Fig. 2 for an illustration.

Algorithm 3. Multivariate attack

Input: $\mathbf{C} \in \mathbb{Z}^{n \times m}$ a basis of $\bar{\mathcal{L}}_{\mathbf{x}}$.

Output: $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \{0,1\}^m$, such that $\mathbf{w}_i \cdot \mathbf{C} = \mathbf{x}_i$ for $i = 1, \ldots, n$.

1: Let $\mathbf{r}_j = ((2 - \delta_{i,k})\mathbf{C}_{ij}\mathbf{C}_{kj})_{1 \le i \le k \le n} \in \mathbb{Z}^{\frac{n^2+n}{2}}$ for $1 \le j \le m$.

2: $\mathbf{E} = \begin{bmatrix} \mathbf{r}_1 \cdots \mathbf{r}_m \\ -\mathbf{C} \end{bmatrix} \in \mathbb{Z}^{\frac{n^2+3n}{2} \times m}$

3: $\mathbf{K} \leftarrow$ Ker $\mathbf{E}$ with $\mathbf{K} = \begin{bmatrix} \mathbf{M} & | & \mathbf{I}_n \end{bmatrix} \in \mathbb{Q}^{n \times \frac{n^2+3n}{2}}$

4: Write $\mathbf{M} = [\tilde{\mathbf{m}}_{ik}]_{1 \le i \le k \le n}$ where $\tilde{\mathbf{m}}_{ik} \in \mathbb{Q}^n$.

5: Let $\mathbf{M}_i \in \mathbb{Q}^{n \times n}$ with $\mathbf{M}_i = [\tilde{\mathbf{m}}_{ik}]_{1 \le k \le n}$, using $\tilde{\mathbf{m}}_{ik} := \tilde{\mathbf{m}}_{ki}$ for $i > k$.

6: $L \leftarrow [\mathbf{I}_n]$

7: **for** $i \leftarrow 1$ to n **do**

8:     $L_2 \leftarrow []$

9:     **for all** $\mathbf{V} \in L$ **do**

10:         **if** rank $\mathbf{V} = 1$ **then**

11:             Append a generator $\mathbf{v}$ of $\mathbf{V}$ to L_2.

12:         **else**

13:             Compute $\mathbf{A}$ such that $\mathbf{V} \cdot \mathbf{M}_i = \mathbf{A} \cdot \mathbf{V}$.

14:             Append all eigenspaces $\mathbf{U}$ of $\mathbf{A}$ to L_2.

15:         **end if**

16:     **end for**

17:     $L \leftarrow L_2$

18: **end for**

19: $X \leftarrow []$

20: **for all** $\mathbf{v} \in L$ **do**

21:     Find $c \ne 0$ such that $\mathbf{x} = c \cdot \mathbf{v} \cdot \mathbf{C} \in \{0,1\}^m$, and append $\mathbf{x}$ to X.
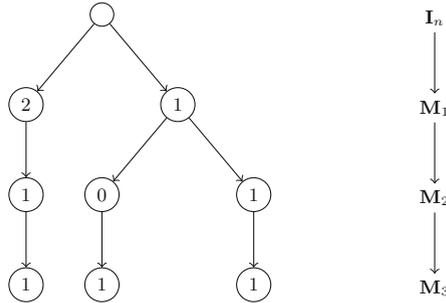
22: **end for**

23: **return** X

Fig. 2. Example of the tree we obtain for $\mathbf{w_1} = (2,1,1), \mathbf{w_2} = (1,0,1), \mathbf{w_3} = (1,1,1)$. The matrix $\mathbf{M}_1$ has an eigenspace of dimension 1 $E_{1,2}$ and one of dimension 2 $E_{1,1}$. At the first iteration we obtain therefore $\mathbf{w_1}$. Then we compute the restriction of $\mathbf{M}_2$ to $E_{1,1}$; this has two distinct eigenvalues 0 and 1, which enables to recover the eigenvectors $\mathbf{w_2}$ and $\mathbf{w_3}$. All the nodes at depth 2 represent dimension one spaces, hence the algorithm terminates.

Analysis and reduction modulo p. Our algorithm is heuristic as we must assume that the matrix $\mathbf{R} \in \mathbb{Z}^{\frac{n^2+n}{2} \times m}$ has rank $(n^2+n)/2$. In our experiments we took $m = (n^2 + 4n)/2$ and this hypothesis was always satisfied. The running time of the algorithm is dominated by the cost of computing the kernel of a matrix $\mathbf{E}$ of dimension $\frac{n^2+3n}{2} \times m$. For $m = (n^2 + 4n)/2$, this requires $\mathcal{O}(n^6)$ arithmetic operations. Thus we have shown:

Lemma 4. *Let $\mathbf{C} \in \mathbb{Z}^{n \times m}$ be an instance of Problem 2 and $\mathbf{R} \in \mathbb{Z}^{\frac{n^2+n}{2} \times m}$ the matrix whose columns are the $\mathbf{r}_i$ constructed as in (13). If $\mathbf{R}$ has rank $\frac{n^2+n}{2}$, then the vectors $\mathbf{x}_i$ can be recovered in $\mathcal{O}(n^6)$ arithmetic operations.*

In practice it is more efficient to work modulo a prime p instead of over $\mathbb{Q}$. Namely Problem 1 is defined over the integers, so we can consider its reduction modulo a prime p:

$$\overline{\mathbf{W}}\mathbf{C} = \overline{\mathbf{X}} \pmod{p}$$

and since $\overline{\mathbf{X}}$ has coefficients in $\{0,1\}$ we obtain a system which is exactly the reduction of (12) modulo p. In particular, we can compute $\mathbf{K} = \ker \mathbf{E}$ modulo p instead of over $\mathbb{Q}$, and also compute the eigenspaces modulo p. Setting $\overline{\mathbf{R}} = \mathbf{R} \bmod p$, if $\overline{\mathbf{R}}$ has rank $\frac{n^2+n}{2}$, then $\mathbf{X}$ can be recovered by $\mathcal{O}(n^6 \cdot \log^2 p)$ bit operations.

Note that we cannot take $p = 2$ as in that case any vector $\mathbf{w}_i$ would be a solution of $\mathbf{w}_i \cdot \mathbf{C} = \mathbf{x}_i \pmod{2}$, since $\mathbf{x}_i \in \{0,1\}^m$. In practice we took $p = 3$ and $m = (n^2 + 4n)/2$, which was sufficient to recover the original vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n$. In that case, the heuristic time complexity is $\mathcal{O}(n^6)$, while the space complexity is $\mathcal{O}(n^4)$. We provide the results of practical experiments in Sect. 7, and the source code in https://pastebin.com/ZFk1qjfP.

5 Improvement of the Algorithm First Step

The first step of our new attack is the same as in [NS99], except that we need to produce $m - n$ orthogonal vectors in $\mathcal{L}_\mathbf{x}^\perp$ from $m = n(n+4)/2$ samples, instead of only $m = 2n$ samples in the original Nguyen-Stern attack. Therefore, we need to produce $n(n+2)/2$ orthogonal vectors in $\mathcal{L}_\mathbf{x}^\perp$, instead of only n. In Sect. 4.1, this required $m/n \simeq n/2$ parallel applications of LLL to compute those $m - n$ vectors in $\mathcal{L}_\mathbf{x}^\perp$, and similarly $n/2$ parallel applications of LLL to compute the orthogonal $\bar{\mathcal{L}}_\mathbf{x} = (\mathcal{L}_\mathbf{x}^\perp)^\perp \in \mathbb{Z}^m$. Overall the heuristic time complexity was $\mathcal{O}(n^{10})$.

In this section, we show that only a single application of LLL (with the same dimension) is required to produce the $m - n$ orthogonal vectors in $\mathcal{L}_\mathbf{x}^\perp$. Namely we show that once the first n orthogonal vectors have been produced, we can very quickly generate the remaining $m - 2n$ other vectors, by size-reducing the original basis vectors with respect to an LLL-reduced submatrix. Similarly a single application of LLL is required to recover a basis of $\bar{\mathcal{L}}_\mathbf{x}$. Eventually the heuristic time complexity of the first step is $\mathcal{O}(n^9)$, as in the original Nguyen-Stern algorithm. This implies that the heuristic complexity of our full algorithm for solving the hidden subset sum problem is also $\mathcal{O}(n^9)$.

5.1   Closest Vector Problem

Size reduction with respect to an LLL-reduced sub-matrix essentially amounts to solving the approximate closest vector problem (CVP) in the corresponding lattice.

Definition 2 (Approximate closest vector problem). *Fix $\gamma > 1$. Given a basis for a lattice $\mathcal{L} \subset \mathbb{Z}^d$ and a vector $\mathbf{t} \in \mathbb{R}^d$, compute $\mathbf{v} \in \mathcal{L}$ such that $\|\mathbf{t} - \mathbf{v}\| \leq \gamma\|\mathbf{t} - \mathbf{u}\|$ for all $\mathbf{u} \in \mathcal{L}$.*

To solve approximate-CVP, Babai's nearest plane method [Bab86] inductively finds a lattice vector close to a vector $\mathbf{t}$, based on a Gram-Schmidt basis. Alternatively, Babai's rounding technique has a worse approximation factor γ but is easier to implement in practice.

Algorithm 4. Babai's rounding method

Input: a basis $\mathbf{b}_1, \ldots, \mathbf{b}_d$ of a lattice $\mathcal{L} \subset \mathbb{Z}^d$. A vector $\mathbf{t} \in \mathbb{Z}^d$.
Output: a vector $\mathbf{v} \in \mathcal{L}$.
 1: Write $\mathbf{t} = \sum_{i=1}^d u_i \mathbf{b}_i$ with $u_i \in \mathbb{R}$.
 2: **return** $\mathbf{v} = \sum_{i=1}^d \lfloor u_i \rceil \mathbf{b}_i$

Theorem 3 (Babai's rounding [Bab86]). *Let $\mathbf{b}_1, \ldots, \mathbf{b}_d$ be an LLL-reduced basis (with respect to the Euclidean norm and with factor $\delta = 3/4$) for a lattice $\mathcal{L} \subset \mathbb{R}^d$. Then the output $\mathbf{v}$ of the Babai rounding method on input $\mathbf{t} \in \mathbb{R}^d$ satisfies $\|\mathbf{t} - \mathbf{v}\| \leq (1 + 2d(9/2)^{d/2})\|\mathbf{t} - \mathbf{u}\|$ for all $\mathbf{u} \in \mathcal{L}$.*

5.2   Generating Orthogonal Vectors in $\mathcal{L}_{\mathbf{x}}^{\perp}$

We start with the computation of the orthogonal vectors in $\mathcal{L}_{\mathbf{x}}^{\perp}$. Consider the large $m \times m$ matrix of vectors orthogonal to $h_1, \ldots, h_m$ modulo M corresponding to the lattice $\mathcal{L}_0$. Our improved technique is based on the fact that once LLL has been applied to the small upper-left $(2n) \times (2n)$ sub-matrix of vectors orthogonal to $(h_1, \ldots, h_{2n})$ modulo M, we do not need to apply LLL anymore to get more orthogonal vectors; namely it suffices to size-reduce the other rows with respect to these $2n$ already LLL-reduced vectors. After size-reduction we obtain short vectors in $\mathcal{L}_0$, and as previously if these vectors are short enough, they are guaranteed to belong to the orthogonal lattice $\mathcal{L}_{\mathbf{x}}^{\perp}$; see Fig. 3 for an illustration. Such size-reduction is much faster than repeatedly applying LLL as in Sect. 4.1. We describe the corresponding algorithm below.
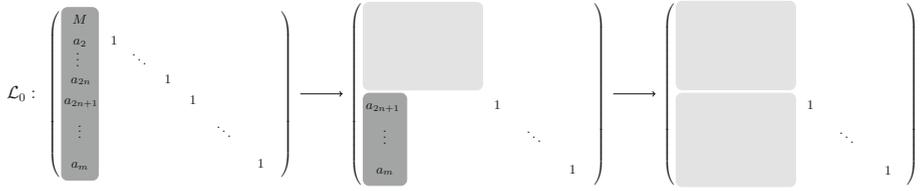
Fig. 3. In the initial basis matrix the components of the first column are big. Then by applying LLL on the $2n \times 2n$ submatrix the corresponding components become small; this already gives n orthogonal vectors in $\mathcal{L}_{\mathbf{x}}^{\perp}$. Then by size-reducing the remaining $m - 2n$ rows, one obtains small components on the $2n$ columns, and therefore $m - 2n$ additional orthogonal vectors. In total we obtain $m - n$ orthogonal vectors.

Algorithm 5. Fast generation of orthogonal vectors

Input: $\mathbf{h} \in \mathbb{Z}^m$, M, n, m.
Output: A generating set of $\mathcal{L}_{\mathbf{x}}^{\perp} \subset \mathbb{Z}^m$.
1: Let $\mathbf{B} \in \mathbb{Z}^{m \times m}$ be a basis of row vectors of the lattice $\mathcal{L}_0$ of vectors orthogonal to $\mathbf{h}$ modulo M, in lower triangular form.
2: Apply LLL to the upper-left $(2n) \times (2n)$ submatrix of $\mathbf{B}$.
3: Let $\mathbf{a}_1, \ldots, \mathbf{a}_{2n} \in \mathbb{Z}^{2n}$ be the $2n$ vectors of the LLL-reduced basis.
4: **for** $i = 2n + 1$ to m **do**
5:     Let $\mathbf{t}_i = [-h_i h_1^{-1}[M] \ 0 \ \cdots \ 0] \in \mathbb{Z}^{2n}$.
6:     Apply Babai's rounding to $\mathbf{t}_i$, with respect to $(\mathbf{a}_1, \ldots, \mathbf{a}_{2n})$. Let $\mathbf{v} \in \mathbb{Z}^{2n}$ be the resulting vector.
7:     Let $\mathbf{a}_i' = [(\mathbf{t}_i - \mathbf{v}) \ 0 \ 1 \ 0] \in \mathbb{Z}^m$ where the 1 component is at index i.
8: **end for**
9: For $1 \leq i \leq n$, extend the vectors $\mathbf{a}_i$ to $\mathbf{a}_i' \in \mathbb{Z}^m$, padding with zeros.
10: Output the n vectors $\mathbf{a}_i'$ for $1 \leq i \leq n$, and the $m - 2n$ vectors $\mathbf{a}_i'$ for $2n + 1 \leq i \leq m$.

The following Lemma shows that under certain conditions on the lattice $\mathcal{L}_{\mathbf{x}}^{\perp}$, Algorithm 5 outputs a generating set of $m - n$ vectors of $\mathcal{L}_{\mathbf{x}}^{\perp}$. More specifically, we have to assume that the lattice $\mathcal{L}_{\mathbf{x}}^{\perp}$ contains short vectors of the form $[\mathbf{c}_i \ 0 \ \ldots \ 1 \ \ldots \ 0]$ with $\mathbf{c}_i \in \mathbb{Z}^{2n}$; this assumption seems to be always verified in practice. We provide the proof in the full version of this paper [CG20].

Lemma 5. *Assume that the lattice $\mathcal{L}_{\mathbf{x}}^{\perp}$ contains n linearly independent vectors of the form $\mathbf{c}_i' = [\mathbf{c}_i \ 0 \cdots 0] \in \mathbb{Z}^m$ for $1 \leq i \leq n$ with $\mathbf{c}_i \in \mathbb{Z}^{2n}$ and $\|\mathbf{c}_i\| \leq B$, and $m - 2n$ vectors of the form $\mathbf{c}_i' = [\mathbf{c}_i \ 0 \ \ldots \ 1 \ \ldots \ 0] \in \mathbb{Z}^m$ where the 1 component is at index i, for $2n + 1 \leq i \leq m$ with $\mathbf{c}_i \in \mathbb{Z}^{2n}$ and $\|\mathbf{c}_i\| \leq B$. Then if $(\gamma B + 1)\sqrt{mn} \leq \lambda_1\left(\Lambda_M^{\perp}(\boldsymbol{\alpha})\right)$ where $\gamma = 1 + 4n(9/2)^n$, Algorithm 5 returns a set of $m - n$ linearly independent vectors in $\mathcal{L}_{\mathbf{x}}^{\perp}$, namely n vectors $\mathbf{a}_i' \in \mathcal{L}_{\mathbf{x}}^{\perp}$ for $1 \leq i \leq n$, and $m - 2n$ vectors $\mathbf{a}_i' \in \mathcal{L}_{\mathbf{x}}^{\perp}$ for $2n + 1 \leq i \leq m$.*

Complexity analysis. Since the approximation factor γ for CVP is similar to the LLL Hermite factor, we use the same modulus size as previously, namely $\log M \simeq 2\iota n^2 + n \cdot \log n$ with $\iota = 0.035$. As in Sect. 3.3 the complexity of the first LLL reduction with L^2 is $\mathcal{O}(n^5 \log^2 M) = \mathcal{O}(n^9)$.

We now consider the size-reductions with Babai's rounding. To apply Babai's rounding we must first invert a $2n \times 2n$ matrix with $\log M$ bits of precision; this has to be done only once, and takes $\mathcal{O}(n^3 \log^2 M) = \mathcal{O}(n^7)$ time. Then for each Babai's rounding we need one vector matrix multiplication, with precision $\log M$ bits. Since the vector has actually a single non-zero component, the complexity is $\mathcal{O}(n \log^2 M) = \mathcal{O}(n^5)$. With $m = \mathcal{O}(n^2)$, the total complexity of size-reduction is therefore $\mathcal{O}(n^7)$. In the full version of this paper [CG20], we describe a further improvement of the size-reduction step, with complexity $\mathcal{O}(n^{20/3})$ instead of $\mathcal{O}(n^7)$.

Overall the heuristic complexity of Algorithm 5 for computing a generating set of $\mathcal{L}_{\mathbf{x}}^{\perp}$ is therefore $\mathcal{O}(n^9)$, instead of $\mathcal{O}(n^{10})$ in Sect. 4.1.

5.3   Computing the Orthogonal of $\mathcal{L}_{\mathbf{x}}^{\perp}$

As in the original [NS99] attack, once we have computed a generating set of the rank $m - n$ lattice $\mathcal{L}_{\mathbf{x}}^{\perp} \subset \mathbb{Z}^m$, we need to compute its orthogonal, with $m = n(n+4)/2$ instead of $m = 2n$. As previously, this will not take significantly more time, because of the structure of the generating set of vectors in $\mathcal{L}_{\mathbf{x}}^{\perp}$. Namely as illustrated in Fig. 4, the matrix defining the $m - n$ orthogonal vectors in $\mathcal{L}_{\mathbf{x}}^{\perp}$ is already almost in Hermite Normal Form (after the first $2n$ components), and therefore once the first $2n$ components of a basis of n vectors of $\bar{\mathcal{L}}_{\mathbf{x}} = (\mathcal{L}_{\mathbf{x}}^{\perp})^{\perp}$ have been computed (with LLL), computing the remaining $m - 2n$ components is straightforward.

More precisely, from Algorithm 5, we obtain a matrix $\mathbf{A} \in \mathbb{Z}^{(m-n) \times m}$ of row vectors generating $\mathcal{L}_{\mathbf{x}}^{\perp}$, of the form:

$$\mathbf{A} = \begin{bmatrix} \mathbf{U} & \\ \mathbf{V} & \mathbf{I}_{m-2n} \end{bmatrix}$$

Fig. 4. Structure of the generating set of $\mathcal{L}_{\mathbf{x}}^{\perp}$.

where $\mathbf{U} \in \mathbb{Z}^{n \times 2n}$ and $\mathbf{V} \in \mathbb{Z}^{(m-2n) \times 2n}$. As in Sect. 3.1, using the LLL-based algorithm from [NS97], we first compute a matrix basis $\mathbf{P} \in \mathbb{Z}^{2n \times n}$ of column vectors orthogonal to the rows of $\mathbf{U}$, that is $\mathbf{U} \cdot \mathbf{P} = 0$. We then compute the matrix:

$$\mathbf{C} = \begin{bmatrix} \mathbf{P} \\ -\mathbf{VP} \end{bmatrix} \in \mathbb{Z}^{m \times n}$$

and we obtain $\mathbf{A} \cdot \mathbf{C} = 0$ as required. Therefore the matrix $\mathbf{C}$ of column vectors is a basis of $\bar{\mathcal{L}}_{\mathbf{x}} = (\mathcal{L}_{\mathbf{x}}^{\perp})^{\perp}$.

6   Cryptographic Applications

In [NS99], the authors showed how to break the fast generator of random pairs $(x, g^x \pmod p)$ from Boyko *et al.* [BPV98], using their algorithm for solving the hidden subset-sum problem. Such generator can be used to speed-up the generation of discrete-log based algorithms with fixed base g, such as Schnorr identification, and Schnorr, ElGamal and DSS signatures. The generator of random pairs $(x, g^x \pmod p)$ works as follows. We consider a prime number p and $g \in \mathbb{Z}_p^*$ of order M.

Preprocessing Step: Take $\alpha_1, \ldots, \alpha_n \leftarrow \mathbb{Z}_M$ and compute $\beta_j = g^{\alpha_j}$ for each $j \in [1, n]$ and store (α_j, β_j).

Pair Generation: To generate a pair $(g, g^x \bmod p)$, randomly generate a subset $S \subseteq [1, n]$ such that $|S| = \kappa$; compute $b = \sum_{j \in S} \alpha_j \bmod M$, if $b = 0$ restart, otherwise compute $B = \prod_{j \in S} \beta_j \bmod p$. Return (b, B).

In [NS99] the authors described a very nice passive attack against the generator used in Schnorr's signatures, based on a variant of the hidden subset-sum problem, called the affine hidden subset-sum problem; the attack is also applicable to ElGamal and DSS signatures. Under this variant, there is an additional secret s, and given $\mathbf{h}, \mathbf{e} \in \mathbb{Z}^m$ one must recover s, the $\mathbf{x}_i$'s and the α_i's such that:

$$\mathbf{h} + s\mathbf{e} = \alpha_1 \mathbf{x}_1 + \alpha_2 \mathbf{x}_2 + \cdots + \alpha_n \mathbf{x}_n \pmod{M}$$

Namely consider the Schnorr's signature scheme. Let q be a prime dividing $p-1$, let $g \in \mathbb{Z}_p$ be a q-th root of unity, and $y = g^{-s} \bmod p$ be the public key. The signer must generate a pair $(k, g^k \bmod p)$ and compute the hash $e = H(\mathsf{mes}, x)$ of the message mes; it then computes $y = k + se \bmod q$; the signature is the pair (y, e). We see that the signatures (y_i, e_i) give us an instance of the affine hidden subset-sum problem above, with $\mathbf{h} = (y_i)$, $\mathbf{e} = (-e_i)$ and $M = q$.

In the full version of this paper [CG20], we recall how to solve the affine hidden subset-sum problem using a variant of the Nguyen-Stern algorithm (in exponential time), and then using our multivariate algorithm (in polynomial time).

7   Implementation Results

We provide in Table 4 the result of practical experiments with our new algorithm; we provide the source code in https://pastebin.com/ZFk1qjfP, based on the L^2 implementation from [fpl16]. We see that for the first step, the most time consuming part is the first application of LLL to the $(2n) \times (2n)$ submatrix of $\mathcal{L}_0$; this first step produces the first n orthogonal vectors. The subsequent size-reduction (SR) produces the remaining $m - n \simeq n^2/2$ orthogonal vectors, and is much faster; for these size-reductions, we apply the technique described in Sect. 5, with the improvement described in the full version of this paper [CG20], with parameter $k = 4$. Finally, the running time of the second LLL to compute the orthogonal of $\mathcal{L}_{\mathbf{x}}^{\perp}$ has running time comparable to the first LLL. As explained previously we use the modulus bitsize $\log M \simeq 2\iota n^2 + n \cdot \log n$ with $\iota = 0.035$.

Table 4. Running time of our new algorithm, for various values of n, under a 3,2 GHz Intel Core i5 processor. We provide the source code and the complete timings in https://pastebin.com/ZFk1qjfP.

n	m	$\log M$	LLL $\mathcal{L}_0$	SR	LLL $\mathcal{L}_{\mathbf{x}}^{\perp}$	Kernel mod 3	Eigenspaces	Total
				Step 1			Step 2	
70	2590	772	3 s	3 s	1 s	8 s	7 s	24 s
90	4230	1151	10 s	8 s	5 s	23 s	17 s	66 s
110	6270	1592	32 s	18 s	11 s	52 s	37 s	153 s
130	8710	2095	87 s	40 s	26 s	112 s	71 s	6 min
150	11550	2659	3 min	70 s	48 s	3 min	122 s	12 min
170	14790	3282	7 min	125 s	81 s	5 min	3 min	20 min
190	18430	3965	23 min	3 min	3 min	9 min	5 min	46 min
220	24640	5099	54 min	7 min	34 min	18 min	8 min	124 min
250	31750	6366	119 min	12 min	65 min	30 min	15 min	245 min

In the second step, we receive as input from Step 1 an LLL-reduced basis of the lattice $\bar{\mathcal{L}}_{\mathbf{x}}$. As described in Algorithm 3 (Step 2), one must first generate a big matrix $\mathbf{E}$ of dimension roughly $n^2/2 \times n^2/2$, on which we compute the kernel $\mathbf{K} = \ker \mathbf{E}$; as explained in Sect. 4.2, this can be done modulo 3. As illustrated in Table 4, computing the kernel is the most time consuming part of Step 2. The computation of the eigenspaces (also modulo 3) to recover the original vectors $\mathbf{x}_i$ is faster.

Comparison with Nguyen-Stern. We compare the two algorithms in Table 5. We see that our polynomial time algorithm enables to solve the hidden subset-sum problem for values of n that are beyond reach for the original Nguyen-Stern attack. Namely our algorithm has heuristic complexity $\mathcal{O}(n^9)$, while the Nguyen-Stern algorithm has heuristic complexity $2^{\Omega(n/\log n)}$. However, we need more samples, namely $m \simeq n^2/2$ samples instead of $m = 2n$.

Table 5. Timing comparison between the Nguyen-Stern algorithm and our algorithm, for various values of n, where m is the number of samples from the generator.

	n	90	110	130	150	170	190	220	250
Nguyen-Stern	m	180	220	260	300	340			
attack [NS99]	time	18 s	50 s	127 s	8 min	447 min	—	—	—
Our attack	m	4230	6270	8710	11550	14790	18430	24640	31750
	time	66 s	153 s	6 min	12 min	20 min	46 min	124 min	245 min

Reducing the number of samples. In the full version of this paper [CG20] we show how to slightly reduce the number of samples m required for our attack, with two different methods; in both cases the attack remains heuristically polynomial time under the condition $m = n^2/2 - \mathcal{O}(n \log n)$. We provide the results of practical experiments in Table 6, showing that in practice the running time grows relatively quickly for only a moderate decrease in the number of samples m.

Table 6. Running time of our new algorithm for $n = 190$, for smaller values of m, for the two methods described in the full version of this paper [CG20].

		Method 1		Method 2	
n	m	Eigenspaces	Total	Eigenspaces	Total
190	17 670	13 min	43 min	2 min	39 min
190	17 480	18 min	57 min	4 min	55 min
190	17 290	29 min	71 min	5 min	50 min
190	17 100	68 min	99 min	8 min	54 min
190	16 910	182 min	217 min	15 min	66 min
190	16 720	—	—	32 min	80 min
190	16 530	—	—	72 min	116 min

References

[Bab86] Babai, L.: On Lovász' lattice reduction and the nearest lattice point problem. Combinatorica **6**(1), 1–13 (1986)

[BNNT11] Brier, É., Naccache, D., Nguyen, P.Q., Tibouchi, M.: Modulus fault attacks against RSA-CRT signatures. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 192–206. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23951-9_13

[BPV98] Boyko, V., Peinado, M., Venkatesan, R.: Speeding up discrete log and factoring based schemes via precomputations. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 221–235. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0054129

[CG20] Coron, J.-S., Gini, A.: A polynomial-time algorithm for solving the hidden subset sum problem. Full version of this paper. Cryptology ePrint Archive, Report 2020/461 (2020). https://eprint.iacr.org/2020/461

[CJL+92] 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)

[CKPS00] Courtois, N., Klimov, A., Patarin, J., Shamir, A.: Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 392–407. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-45539-6_27

[CLO05] Cox, D.A., Little, J., Oshea, D.: Using Algebraic Geometry. Springer, New York (2005). https://doi.org/10.1007/b138611

[CLT13] Coron, J.-S., Lepoint, T., Tibouchi, M.: Practical multilinear maps over the integers. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 476–493. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40041-4_26

[CN11] Chen, Y., Nguyen, P.Q.: BKZ 2.0: better lattice security estimates. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 1–20. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25385-0_1

[CN19] Coron, J.-S., Notarnicola, L.: Cryptanalysis of CLT13 multilinear maps with independent slots. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019, Part II. LNCS, vol. 11922, pp. 356–385. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-34621-8_13

[CNT10] Coron, J.-S., Naccache, D., Tibouchi, M.: Fault attacks against EMV signatures. In: Pieprzyk, J. (ed.) CT-RSA 2010. LNCS, vol. 5985, pp. 208–220. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-11925-5_15

[CP19] Coron, J.-S., Pereira, H.V.L.: On Kilian's randomization of multilinear map encodings. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019, Part II. LNCS, vol. 11922, pp. 325–355. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-34621-8_12

[FLLT15] Fouque, P.-A., Lee, M.S., Lepoint, T., Tibouchi, M.: Cryptanalysis of the Co-ACD assumption. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015, Part I. LNCS, vol. 9215, pp. 561–580. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47989-6_27

[fpl16] The FPLLL development team. FPLLL, a lattice reduction library (2016). https://github.com/fplll/fplll

[HPS11] Hanrot, G., Pujol, X., Stehlé, D.: Analyzing blockwise lattice algorithms using dynamical systems. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 447–464. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22792-9_25

[LLL82] Lenstra, A.K., Lenstra, H.W., Lovasz, L.: Factoring polynomials with rational coefficients. Math. Ann. **261**, 515–534 (1982). https://doi.org/10.1007/BF01457454

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

[LT15] Lepoint, T., Tibouchi, M.: Cryptanalysis of a (somewhat) additively homomorphic encryption scheme used in PIR. In: Brenner, M., Christin, N., Johnson, B., Rohloff, K. (eds.) FC 2015. LNCS, vol. 8976, pp. 184–193. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48051-9_14

[NS97]   Nguyen, P., Stern, J.: Merkle-Hellman revisited: a cryptanalysis of the Qu-Vanstone cryptosystem based on group factorizations. In: Kaliski, B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 198–212. Springer, Heidelberg (1997). https://doi.org/10.1007/BFb0052236

[NS98a] Nguyen, P., Stern, J.: The Béguin-Quisquater server-aided RSA protocol from Crypto '95 is not secure. In: Ohta, K., Pei, D. (eds.) ASIACRYPT 1998. LNCS, vol. 1514, pp. 372–379. Springer, Heidelberg (1998). https://doi.org/10.1007/3-540-49649-1_29

[NS98b] Nguyen, P., Stern, J.: Cryptanalysis of the Ajtai-Dwork cryptosystem. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 223–242. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0055731

[NS99]   Nguyen, P., Stern, J.: The hardness of the hidden subset sum problem and its cryptographic implications. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 31–46. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_3

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

[NSS04] Naccache, D., Smart, N.P., Stern, J.: Projective coordinates leak. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 257–267. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24676-3_16

[Sag19] The Sage Developers. Sagemath, the Sage Mathematics Software System (Version 8.9) (2019). https://www.sagemath.org

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

[Sho]   Shoup, V.: Number theory C++ library (NTL) version 3.6. http://www.shoup.net/ntl/

[vDGHV10] van Dijk, M., Gentry, C., Halevi, S., Vaikuntanathan, V.: Fully homomorphic encryption over the integers. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 24–43. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13190-5_2