UNIVERSITÉ DU LUXEMBOURG

PhD-FSTM-2022-074
The Faculty of Science, Technology and Medicine

DISSERTATION

Defence held on 07/07/2022 in Esch-sur-Alzette

to obtain the degree of

DOCTEUR DE L'UNIVERSITÉ DU LUXEMBOURG

EN INFORMATIQUE

by

Agnese GINI
Born on 20 April 1992 in Pontedera, (Italy)

ON THE HARDNESS OF THE HIDDEN SUBSET SUM PROBLEM: ALGEBRAIC AND STATISTICAL ATTACKS

Dissertation defence committee

Dr Jean-Sébastien CORON, dissertation supervisor
Professor, Université du Luxembourg

Dr Gabor WIESE, Chairman
Professor, Université du Luxembourg

Dr Frederik VERCAUTEREN, Vice Chairman
Professor, KU Leuven

Dr Phong NGUYEN
Professor, ENS PSL

Dr Damien STEHLÉ
Professor, ENS Lyon

Introduction

In the last decades, the massive development of the internet of things (IoT) has provoked an increasing interest in efficient implementations of cryptographic primitives. The will to integrate cryptographic protocols into small devices for daily usage has resulted in an increasing demand of reducing memory and time consumption. Therefore, several methods have been developed for this purpose. However, it has been observed that embedding these techniques in signatures, encryption, *etc.*, quite often introduces security issues. Hence, an accurate cryptanalysis of such methods has become actually relevant.

The fast generator of random discrete-log pairs $(x, g^x \pmod p)$ by Boyko, Peinado and Venkatesan was indeed presented at Eurocrypt '98 [BPV98] for the purpose of speeding up computations in small devices, such as smart cards. Given a hidden set of precomputed pairs $\{(\alpha_i, g^{\alpha_i} \pmod p)\}$ the BPV algorithm produces new pairs by combining random subsets. Due to its simple description, the BPV generator has been later enhanced, revisited and used for various applications such as server delegation [NSS01, CLV21], wireless network systems [ABC+17, OY17], and drones [MV22, ZJL+21, OY18, LYZ+20].

The cryptanalysis of this method is based on the *hidden subset sum problem* (HSSP), a variant of the traditional subset sum problem where the n weights are hidden.

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

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

Given the modulus Q *and the* sample vector $\mathbf{h}$*, 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.*

Despite the simple formulation of HSSP, its hardness has not been completely understood yet. Indeed, for certain choices of parameters the problem is at least as hard as the subset sum problem, while its complexity is still unexplored for other sets of parameters. The purpose of this thesis is to analyse the hardness of the hidden subset sum problem and some of its

extensions, having the task of identifying families of easy and hard instances both theoretical and practical relevance. Specifically, we discuss methods for solving these problems, involving tools from various mathematical areas and reductions to special instances of well-known problems, such as finding short vectors of a lattice, solving a multivariate polynomial system, and disclosing a hidden statistical distribution.

The hardness of the hidden subset sum problem was first discussed by Nguyen and Stern in their paper [NS99] presented at Crypto'99, where sets of parameters suitable for cryptography were suggested and the first lattice based attack described. In [NSS01] the same authors together with Shparlinski also proved sufficient conditions under which the distribution of the output of the BPV generator is indistinguishable from the uniform distribution. These two works however do not identify necessary conditions for the hardness. Indeed, although the Nguyen-Stern algorithm works quite well in practice for moderate values of n, we have observed that its complexity is likely to be exponential in n. This is because the final step recovers a very short basis of an n-dimensional lattice, and this takes time exponential in n, since the BKZ reduction algorithm with increasingly large block-sizes has to be applied. Nevertheless, the Nguyen-Stern algorithm contains several interesting ideas and served as a basis for our contributions, which aim to further develop the understanding of the hardness of the hidden subset problem and its variants.

The main contribution of this thesis is the development of polynomial-time algorithms for solving the hidden subset sum problem, improving upon the existing algorithms that are exponential-time. Indeed, we describe two new attacks and we identify parameters for which they work in polynomial time and space. Furthermore, we introduce the *hidden linear combination problem* (HLCP), which is an extension of the HSSP where the coefficients of the $\mathbf{x}_i$'s belong to a larger range $\{0, \ldots, B\}$ for $B \in \mathbb{N}^+$, instead of $\{0, 1\}$ only. This problem naturally arises from extending the BPV generator (EBPV), as proposed in [NSS01], by including small exponentiation. Namely, given a hidden set of precomputed pairs $\{(\alpha_i, g^{\alpha_i} \pmod p)\}$, the authors suggest to produce new pairs by performing linear combinations of the exponents: $x = \sum_{i=1}^{n} \alpha_i \cdot x_i \bmod q$ and so

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

where $x_i \in \{0, \ldots, B\}$. Therefore, as the last contribution of this thesis, we briefly discuss the hardness of the HLCP, analysing the adaptation of the known attacks to solve the HSSP to HLCP.

Road map and main contributions of this thesis

This thesis comprises our results included in the works *A polynomial-time algorithm for solving the hidden subset sum problem* [CG20a] and *Provably solving the hidden subset sum problem via statistical learning* [CG22], both joint with Jean-Sébastien Coron and presented at Crypto2020 and MathCrypt2021, respectively.

The content of these papers is here revised, enriched and complemented with novel contributions. This entire thesis is constructed as a framework for a coherent and consistent analysis of the hidden subset sum problem and its variants.

Background Chapter 1 is a collection of preliminaries supporting the main content of the thesis. In particular, the BPV and EBPV generators are described in Section 1.6.

Introduction to the hidden subset sum problem In Chapter 2 we introduce the hidden subset sum problem in detail, and we provide more precise explanation. In particular, in Section 2.1, we specify the notion of *random* instance, identifying the main parameters of the problem and defining two main families of hidden subset sums. More specifically, each component of the vector $\mathbf{h}$ in (I) is a subset sum of the hidden weights $\alpha_1, \ldots, \alpha_n$; if the m coordinates are sampled as classic subset sums, then each subset has the same probability to appear, while only subsets of prescribed cardinality κ are selected if the coordinates are generated by the BPV generator. Therefore, we define the two families of random instances of the hidden subset sum problem $\mathrm{HSSP}_n(m, Q)$ (Definition 2.1) and $\mathrm{HSSP}_n^{\kappa}(m, Q)$ (Definition 2.2), as those naturally descending from either the classic subset sum or the BPV generator, respectively. Since depending on the distribution of the binary vectors the problem inherits different properties, we will observe that making a distinction is relevant. In Section 2.1 we establish some additional features, useful to characterise subfamilies of parameters for which we expect the problem to be hard. Then, we show that for some choices of parameters the distribution of the sample vector $\mathbf{h}$ is close to be indistinguishable from the uniform distribution over $\mathbb{Z}_Q^m$, and we can not expect to find a polynomial time solver. Our work will conversely imply that for certain complementary choices this is actually possible. Namely, through our analysis of Nguyen-Stern's and our algorithms, in the successive chapters we will essentially determine a subset of instances that are easy *on average*. Furthermore, Section 2.2 includes an overall description of the algorithms for solving the hidden subset problem later discussed in the thesis.

The Nguyen-Stern algorithm Chapter 3 is devoted to our analysis of the lattice based algorithm for solving the hidden subset sum problem described by Nguyen and Stern in [NS99]. The authors propose to divide the algorithm for solving the problem in two parts: the first step consists in an algorithm for disclosing a hidden lattice containing the binary vectors, while the second step is essentially a binary vectors' search inside of such a lattice. In Section 3.1, we illustrate the algorithm according to the version we formalised in [CG20a]. Then in Section 3.2 we provide our analysis, which is an extension of that one presented in [CG20a], taking into account our novel framework, introduced in Chapter 2. The original paper of Nguyen and Stern only included a heuristic analysis of the first step, which is performed by an orthogonal lattice attack using LLL. Adapting and enhancing that analysis, we are able to find a rigorous condition under which the hidden lattice can be recovered. In particular, we derive a rigorous lower bound for the bitsize of the modulus Q, as a function of the number of weights n, above which the orthogonal lattice attack

succeeds in polynomial time. We also compute further heuristic bounds, inferable from our analysis.

Then, we observe that the bottleneck of Nguyen-Stern attack resides actually in the second step, as they employ a lattice reduction strategy for retrieving the binary vectors, and for large values of n this strategy is impractical. Specifically, in Section 3.2.2 we provide some heuristic arguments supporting that the algorithm should use BKZ with block-size increasing almost linearly with n to succeed. This would imply a complexity exponential in n. Moreover, in this chapter we also explain some differences between $\mathrm{HSSP}_n(m, Q)$ and $\mathrm{HSSP}_n^{\kappa}(m, Q)$. In particular, for the second step we address the problem separately, showing that for the latter family retrieving the correct basis is also not guaranteed.

Our algorithm for solving HSSP in polynomial time Our main contribution of Chapter 4 is a new approach for solving the hidden subset sum problem in polynomial time. Our analysis of Chapter 3 demonstrates that the orthogonal lattice attack actually works well; therefore, our algorithm maintains the two step's structure, but we propose an alternative method for recovering the binary vectors. Our idea is to reduce the problem of disclosing the binary vectors to solving a *multivariate polynomial system*. This algorithm heuristically requires $m \approx n^2$ samples and solves the problem with full asymptotic complexity $\mathcal{O}(n^9)$. As in the analysis Nguyen-Stern algorithm, we observe that there are some differences between random instances of $\mathrm{HSSP}_n(m, Q)$ and $\mathrm{HSSP}_n^{\kappa}(m, Q)$, too. Namely, we show that in the latter case additional binary vectors naturally arise, and this has an impact especially on the second step. Therefore, in Section 4.2.2 we describe two distinct strategies for our multivariate approach, one for each family of hidden subset sum problems.

In order to be able to recover the binary vectors in polynomial time, all our new methods require large values of m, compared to the original Nguyen-Stern attack that uses $m = 2n$. Thus, informally, we operate a trade-off between time and memory of the binary vectors' search. For this reason, naively applying the orthogonal lattice attack, as described in Chapter 3, is not convenient, since the running time of the algorithm inconveniently increases according to the value of m. Thus, in Section 4.1 we discuss new improvements for the orthogonal lattice attack having two main consequences: first, they allow us to revise the lower bound on the size of the modulus Q, releasing its dependence on the number of samples, and secondly they make the attack feasible in practice.

Finally, in Section 4.3 we address the problem of distinguishing the target vectors for instances of $\mathrm{HSSP}_n^{\kappa}(m, Q)$ from the additional binary vectors in the considered lattice. Indeed, we present an algorithm fulfilling this purpose via exploiting statistical properties of the problem.

The improved orthogonal lattice attack and multivariate approach for HSSP_n are included in [CG20a].

Our statistical algorithm for $\mathbf{HSSP}_n$ We describe in Chapter 5 another method for solving HSSP_n based on a *statistical learning technique*. Specifically, we reduce the problem of retrieving the binary vectors to solving an instance of the *hidden parallelepiped problem*

introduced by Nguyen and Regev in [NR09]. While the multivariate attack is heuristic, this statistical method is proven to be successful in polynomial time with constant probability. Furthermore, in Section 7.2 we will also show that this technique can be adapted for heuristically solving HSSP_n with fewer samples, *e.g.* for $m \simeq n^2$ it recovers the $\mathbf{x}_i$'s in less than a minute for n up to 220. This contribution is included in [CG22].

Variants of the hidden subset sum problem Chapter 6 is devoted to the study of two variants of the hidden subset sum problem: the *affine hidden subset sum problem* and *hidden linear combination problem*. In the affine variant, the main difference is that the sample vector $\mathbf{h}$ in (II) includes an extra shift of an additional vector $s \cdot \mathbf{e}$, where $\mathbf{e}$ is a known vector having components with bounded size and s is a hidden coefficient. While, the hidden linear combination problem is a natural extension of the HSSP, where the coefficients of the $\mathbf{x}_i$'s belong to a larger range $\{0, \dots, B\}$ for a given $B \in \mathbb{N}^+$ and the sample vector $\mathbf{h}$ is obtained as a linear combination of these $\mathbf{x}_i$'s.

Definition II (Affine Hidden Subset Sum Problem). *Let Q be an integer, and let $s, \alpha_1, \dots, \alpha_n$ be random integers in $\mathbb{Z}_Q$. Let $\mathbf{x}_1, \dots, \mathbf{x}_n \in \mathbb{Z}^m$ be vectors with components in $\{0, 1\}$ and $\mathbf{e} = (e_1, \dots, e_m) \in \mathbb{Z}^m$ be a vector with components in $[0, 2^t[$ for $t \in \mathbb{N}$. Let $\mathbf{h} = (h_1, \dots, h_m) \in \mathbb{Z}^m$ satisfying:*

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

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

Definition III (Hidden Linear Combination Problem). *Let Q be a positive integer, and let $\alpha_1, \dots, \alpha_n$ be integers in $\mathbb{Z}_Q$. Let $\mathbf{x}_1, \dots, \mathbf{x}_n \in \mathbb{Z}^m$ be vectors with components in $\{0, \dots, B\}$. Let $\mathbf{h} = (h_1, \dots, h_m) \in \mathbb{Z}^m$ satisfying:*

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

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

In the first section of this chapter, we briefly analyse the affine hidden subset problem, which is a variant introduced in [NS99]. Nguyen and Stern observed that in certain cases the affine variant is actually the correct problem to consider and explained how to modify their attack to solve this variant, too. Therefore, as done for the HSSP, we illustrate in Section 6.1 a rigorous analysis of the algorithm, and we place this problem in the framework we created.
The latter part of the chapter is about the hidden linear combination problem. We defined this problem in [CG22] and we are not aware of any similar definition in previous literature. The hardness of HLCP is linked to the cryptanalysis of the EBPV generator. Recall that this is obtained from the BPV generator by including small exponentiation [NSS01], namely

any generated h is a combination of the form $\alpha_1 x_1 + \cdots + \alpha_n x_n$ where $x_i \in \{0, \ldots, B\}$ and exactly κ out of n of them are non-zero. Indeed, the components of $\mathbf{h}$ in (III) are obtained as positive linear combinations of that form instead of subset sums. Thus, in Section 6.3 we discuss the notion of *random* instance, extending the model established for the HSSP, and so we define two families of problems $\mathrm{HLCP}^{\kappa}_{n,B}(m, Q)$ and $\mathrm{HLCP}_{n,B}(m, Q)$ that extend $\mathrm{HSSP}^{\kappa}_{n}(m, Q)$ and $\mathrm{HSSP}_{n}(m, Q)$, respectively. Namely, for the first family we consider combinations with a prescribed number of zeros, while we consider positive linear combinations with uniformly distributed coefficients for the second one.
The addition of the parameter B modifies the hardness of the problem, causing the impracticability of certain attacks. For instance, extending our multivariate attack becomes infeasible already for quite small $B > 1$. In Section 6.4 we extend the Nguyen-Stern attack to this new problem. We show that also in this case the orthogonal lattice attack works quite well and we are able to provide a rigorous condition for its success in polynomial time, as done for HSSP. Moreover, our heuristic analysis of the application of BKZ for recovering the bounded vectors $\mathbf{x}_i$ suggests that the running-time is $2^{\Omega(n)} \cdot \log^{\mathcal{O}(1)} B$. The behaviour of the extended Nguyen-Stern attack is thus coherent with the case $B = 1$.
Our last contribution regarding the hidden linear combination problem is a heuristic variant of the statistical attack for the HSSP presented in Chapter 5. We first run the orthogonal lattice attack and then instead of producing a continuous distribution and applying the Nguyen-Regev algorithm, we use the FastICA algorithm [HO97]. Indeed, we observed that the Nguyen-Regev algorithm can be actually seen itself as an instantiation of the FastICA algorithm, *i.e.* an algorithm for solving the *signal source separation problem*, when kurtosis is chosen as a cost function. The main advantage of using FastICA is that we can partially release the algorithm from the hypothesis on the underlying distribution and process the discrete samples directly. Hence, we can expect a better condition on m. A heuristic analysis of this algorithm indeed suggests that the asymptotic complexity is $\mathsf{poly}(n, B)$.
Finally, we show that when $B > 1$ there are choices of n, B and κ for which the algorithm solves $\mathrm{HSSP}^{\kappa}_{n}(m, Q)$, too. Our algorithm based on statistical learning is discussed in Section 6.5 (a preliminary version appears in [CG22]).

Other contributions

We summarise here two other works developed during the doctoral studies, not included in this thesis.

- *Improved cryptanalysis of the AJPS Mersenne based cryptosystem* joint with Jean-Sébastien Coron presented at NutMiC2019 [CG20b].
  At Crypto 2018, Aggarwal, Joux, Prakash and Santha (AJPS) described a new public-key encryption scheme based on arithmetic modulo Mersenne numbers [AJPS18]. The arithmetic modulo p, *i.e.* a prime integer of the form $p = 2^n - 1$, has good properties. Specifically, one can define a correspondence between integers modulo p and binary strings of length

n. Then, the *Hamming weight* of a number mod p is the Hamming weight of the unique binary string associated to it. In the earliest version of their work, the authors presented a public-key encryption scheme (AJPS-1) somewhat similar to the NTRU cryptosystem, but based on a new assumption: the *Mersenne Low Hamming Ratio Assumption*. The security of such a system relies on the assumption that, given $H = F/G \mod p$, where the binary representation of F and G modulo p has low Hamming weight, then H looks pseudorandom. Namely, that it is hard to distinguish H from a random integer modulo p. The authors claimed at first that the known lattice attacks against NTRU would not apply. However, Beunardeau et al. [BCGN17] described a lattice-based attack against the first AJPS proposal, whose complexity is $\mathcal{O}(2^{2h})$, where h is the Hamming weight of F and G. This attack was further analysed in [dBDJdW18]; where the authors also described a Meet-in-the-Middle attack against AJPS-1 based on locality-sensitive hash functions to obtain collisions. It was shown that the lattice attack from [BCGN17] is actually more efficient. The first proposal AJPS-1 allows to encrypt only a single bit at a time; while in a later version of the article, published at Crypto 2018 [AJPS18], Aggarwal et al. described a variant (AJPS-2) that encrypts many bits at a time. Our contribution is a variant of the Beunardeau *et al.* attack against AJPS-2, with improved complexity $\mathcal{O}(2^{1.75h})$ instead of $\mathcal{O}(2^{2h})$. Specifically, our attack only breaks the indistinguishability of ciphertexts, rather than recovering the private-key.

- *On the weightwise nonlinearity of weightwise perfectly balanced functions* joint with Pierrick Méaux currently under journal submission [GM22].
  In the stream cipher family FLIP [MJSC16] for hybrid homomorphic encryption, Boolean functions are applied on input ranging only on a subset of $\mathbb{F}_2^n$, rather than the full space. Therefore, for a correct understanding of the property of such new construction, some of the most relevant cryptographic criteria of Boolean functions had to be adapted. This task was first addressed by Carlet *et al.* in [CMR17]. The main purpose of this work is to improve the general understanding of *weightwise perfectly balanced* (WPB) functions, which are those Boolean functions balanced on each set $E_{k,n} = \{x \in \mathbb{F}_2^n \,|\, hw(x) = k\}$ for $1 \leq k \leq n-1$. Specifically, we perform a general study of the *weightwise nonlinearity* of WPB functions, *i.e.* for a function f the minimum distance between its restriction to $E_{k,n}$ and affine functions, restricted to the same set. For this purpose, we exploit the connections with other concepts such as spherically punctured Reed Muller codes, zeroes of Krawtchouk polynomials, and weightwise affine functions. We investigate the minimal and maximal values of the weightwise nonlinearity of WPB functions, the distribution of these values, and we finally consider new constructions.

Contents

1

Background

1.1 Introduction to lattices

In this section, we recall the basics about lattices, following [Cas97], [Mar13] and [NV09]. More precisely, after recalling the generic notion of basis and some related facts, we introduce integer lattices, the concepts of orthogonal lattice and completion, and useful properties. Then we recall the notions of minima and reduced basis with the main bounds.

1.1.1 Lattices and bases

Definition 1.1. *Let* $\mathbf{b}_1, \ldots, \mathbf{b}_d \in \mathbb{R}^m$ *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 the matrix $\mathbf{B}$ is a *base matrix* for the lattice generated by its rows $\mathbf{b}_1, \ldots, \mathbf{b}_d$ and in that case we have

$$\mathcal{L}(\mathbf{B}) \coloneqq \mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_d) = \left\{ \mathbf{a} \cdot \mathbf{B} \mid \mathbf{a} \in \mathbb{Z}^d \right\}.$$

We say that two bases $\mathbf{B}$ and $\mathbf{B}'$ are *equivalent* if they generate the same lattice $\mathcal{L}(\mathbf{B}) = \mathcal{L}(\mathbf{B}')$.

Lemma 1.2. *Two basis* $\mathbf{b}_1, \ldots, \mathbf{b}_d$ *and* $\mathbf{b}'_1, \ldots, \mathbf{b}'_d$ *are equivalent if and only if there exists a unimodular matrix* $\mathbf{U} \in \mathrm{GL}_d(\mathbb{Z})$ *such that* $\mathbf{UB} = \mathbf{B}'$.

Given any basis $\mathbf{B}$ we can consider its Gram-determinant $d(\mathbf{B}) = \sqrt{\det(\mathbf{B}\mathbf{B}^\intercal)}$. The previous Lemma implies that this number is invariant under base change, *i.e.* for any two basis $\mathbf{B}, \mathbf{B}'$ we have $d(\mathbf{B}) = d(\mathbf{B}')$.

Definition 1.3. *The* determinant *of a lattice $\mathcal{L}$ is the Gram-determinant of any of its bases* $\mathbf{B}$*, namely* $\det(\mathcal{L}) = d(\mathbf{B})$.

Let $\mathbf{b}_1, \ldots, \mathbf{b}_d$ be an ordered basis of a lattice $\mathcal{L}$ and $\mathbf{b}_1^*, \ldots, \mathbf{b}_d^*$ its Gram-Schmidt orthogonalization. Letting $\mathbf{Q}$ be such that $\mathbf{B} = \mathbf{Q}\mathbf{B}^*$, this transformation is orthogonal i.e. $\mathbf{Q}\mathbf{Q}^\intercal = \mathbf{I}_d$. Thus,

$$d(\mathbf{B}) = \sqrt{\det((\mathbf{Q}\mathbf{B}^*)(\mathbf{Q}\mathbf{B}^*)^\intercal)} = \sqrt{\det(\mathbf{Q})\det(\mathbf{B}^*(\mathbf{B}^*)^\intercal)\det(\mathbf{Q}^\intercal)} = d(\mathbf{B}^*)$$

Since the $\mathbf{b}_i^*$ are orthogonal, $\det(\mathcal{L}) = d(\mathbf{B}^*) = \prod_{i=1}^d \|\mathbf{b}_i^*\|$, where $\| \cdot \|$ is the Euclidean norm. Moreover, since $\|\mathbf{b}_i^*\| \leq \|\mathbf{b}_i\|$, this implies Hadamard's inequality:

$$\det(\mathcal{L}) \leq \prod_{i=1}^{d} \|\mathbf{b}_i\|.$$

The *dimension* or *rank* of lattice $\dim(\mathcal{L})$ is the dimension as vector space of

$$E_{\mathcal{L}} \coloneqq \mathrm{Span}_{\mathbb{R}}(\mathcal{L}),$$

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 $\mathcal{M}$. A sublattice is a subgroup, if $\dim(\mathcal{L}) = \dim(\mathcal{M})$ the index $[\mathcal{L}\colon \mathcal{M}]$ is finite and $[\mathcal{L}\colon \mathcal{M}] = \det(\mathcal{M})/\det(\mathcal{L})$; in this case we say that $\mathcal{M}$ is a *full-rank sublattice*. Observe that it must be that $\det(\mathcal{L}) \leq \det(\mathcal{M})$.

If $\mathcal{L}(\mathbf{M}) \subseteq \mathcal{L}(\mathbf{B})$ is a sublattice, there exists a matrix $\mathbf{Q}$ with integers coefficients such that $\mathbf{Q}\mathbf{B} = \mathbf{M}$. Indeed the rows of $\mathbf{Q}$ are the coordinates of the given basis of $\mathcal{L}(\mathbf{M})$ with respect to $\mathbf{B}$, since every point of the sublattice is also a point of $\mathcal{L}(\mathbf{B})$. If $\mathcal{L}(\mathbf{M})$ is a full-rank sublattice, $\mathbf{Q}$ must be invertible and $\mathbf{Q}^{-1} = \frac{1}{\det(\mathbf{Q})}\widehat{\mathbf{Q}}$ with $\widehat{\mathbf{Q}} \in \mathrm{M}_n(\mathbb{Z})$ the adjunct of $\mathbf{Q}$ and $\det(\mathbf{Q}) \in \mathbb{Z}$. This implies that $\det(\mathbf{Q}) \cdot \mathbf{B} = \widehat{\mathbf{Q}}\mathbf{M}$, so

$$\det(\mathbf{Q}) \cdot \mathcal{L}(\mathbf{B}) \subseteq \mathcal{L}(\mathbf{M}) \subseteq \mathcal{L}(\mathbf{B}) \tag{1.1}$$

where we denote by $\lambda \cdot \mathcal{L}$ the lattice of $\lambda \cdot \mathbf{v}$ for $\mathbf{v} \in \mathcal{L}$.

In the case of sublattices we can have a basis of special shape.

Theorem 1.4 ([Cas97, Theorem I.B]). *Let $\mathcal{L} \subseteq \mathbb{R}^m$ be a lattice of dimension d and $\mathbf{m}_1, \ldots, \mathbf{m}_d \in \mathcal{L}$ be linearly independent vectors. Then there exists a $d \times d$ lower triangular integral matrix $\mathbf{V} = (v_{i,j})$ and $\mathbf{b}_1, \ldots, \mathbf{b}_d$ a basis of $\mathcal{L}$ such that $\mathbf{m}_i = \sum_{j=1}^{i} v_{i,j}\mathbf{b}_j$.*

Corollary 1.5. *Let $\mathcal{L} \subseteq \mathbb{R}^m$ be a lattice of dimension d and $\mathcal{M}$ a sublattice of dimension r. A basis $\mathbf{m}_1, \ldots, \mathbf{m}_r$ of $\mathcal{M}$ can be completed to a basis $\mathbf{m}_1, \ldots, \mathbf{m}_d$ of $\mathcal{L}$ if and only if for any $\mathbf{c} \in \mathcal{L}$ such that*

$$\mathbf{c} = u_1\mathbf{m}_1 + \cdots + u_r\mathbf{m}_r$$

the real coefficients $u_1, \dots, u_r$ are integral.

Proof. The condition is clearly necessary. To prove that it is sufficient, consider some vectors $\mathbf{n}_{r+1}, \dots, \mathbf{n}_d \in \mathcal{L}$ such that the lattice $\mathcal{M} + \mathcal{L}(\mathbf{n}_{r+1}, \dots, \mathbf{n}_d)$ is full-rank in $\mathcal{L}$. By Theorem 1.4 there exists $\mathbf{b}_1, \dots, \mathbf{b}_d$ a basis of $\mathcal{L}$ such that $\mathbf{m}_i = \sum_{j=1}^{i} v_{i,j}\mathbf{b}_j$. We can also suppose that $v_{i,i} > 0$ and $0 \leq v_{i,j} < v_{i,i}$, since for the first condition it is enough to multiply both $\mathbf{b}_i$ and $\mathbf{m}_i$ by -1 and for the second it is enough to inter-reduce the basis.

Now, $\mathbf{b}_1, \dots, \mathbf{b}_r$ satisfy a relation such that $\mathbf{b}_i = \sum_{j=1}^{r} u_{i,j}\mathbf{m}_j$ with $u_{i,j} \in \mathbb{R}$. Then by hypothesis we must have $\mathbf{b}_i = \sum_{j=1}^{r} u_{i,j}\mathbf{m}_j$ with $u_{i,j} \in \mathbb{Z}$. Since $u_{i,i} = 1/v_{i,i}$ the diagonal elements must be one. This also means that $v_{i,j} = 0$ for $i \neq j$ because of the special form of the matrix $\mathbf{V}$. Thus, $\mathbf{b}_i = \mathbf{m}_i$ for $i \leq r$. Then $\mathbf{m}_i = \mathbf{b}_i$ for $i > r$ complete the $\mathbf{m}_1, \dots, \mathbf{m}_r$ to a basis of $\mathcal{L}$.

□

Integer lattices

Sublattices of $\mathbb{Z}^m$ have peculiar properties. These are called *integer lattices*, and in this thesis we will generally consider this type, only. Therefore, for the sake of simplicity, in the following by *lattice* we always mean integer lattice.

Informally, lattices can be seen as a discrete analogue of vector spaces. So, one can consider the structure given by the standard scalar product $\langle\ ,\ \rangle$ of $\mathbb{R}^m$. Thus, it is natural to define the notion of duality:

Definition 1.6. *Let $\mathcal{L} \subseteq \mathbb{Z}^m$ be a lattice. Its* dual lattice *is*

$$\mathcal{L}^\vee \coloneqq \{\mathbf{v} \in E_\mathcal{L} \ \mid\ \forall \mathbf{b} \in \mathcal{L},\ \langle \mathbf{v}, \mathbf{b}\rangle \in \mathbb{Z}\}\,.$$

Observe that $\mathbb{Z}^m = (\mathbb{Z}^m)^\vee$. Consider now $\mathbf{b}_1, \dots, \mathbf{b}_d$ a basis of a lattice $\mathcal{L}$. Its dual basis $\mathbf{v}_1, \dots, \mathbf{v}_d$ with respect to the standard product must be a basis of $\mathcal{L}^\vee$, since $\langle \mathbf{v}_j, \mathbf{b}_i\rangle = 1$ if $i = j$ and 0 otherwise, and therefore $\mathbf{B}\mathbf{V}^\intercal = \mathbf{I}_d$. From the uniqueness we get $\mathbf{V} = (\mathbf{B}^\intercal\mathbf{B})^{-1}\mathbf{B}$. This also implies that

$$\det(\mathcal{L}) \cdot \det(\mathcal{L}^\vee) = 1.$$

Recall that given $V \subseteq \mathbb{R}^m$ a vector space, we can always define its orthogonal as $V^\perp = \{\mathbf{v} \in \mathbb{R}^m \ \mid\ \forall \mathbf{w} \in V,\ \langle \mathbf{v}, \mathbf{w}\rangle = 0\}$. The dual lattice contains a set of elements in $\mathbb{Z}^m$ which are orthogonal to all the lattice points, more precisely:

Definition 1.7. *Let $\mathcal{L} \subseteq \mathbb{Z}^m$ be a lattice. Its* orthogonal lattice *is*

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

Note that (1.1) implies that the orthogonal of a lattice $\mathcal{L}$ is the same as the orthogonal of a full-rank sublattice $\mathcal{M} \subset \mathcal{L}$.

We observed that it is not always true that a basis of a sublattice can be completed to a basis of a lattice; see Corollary 1.5. However, using the orthogonal we can always find a superlattice of the same dimension which has such property. More precisely, we define the *completion* of a lattice $\mathcal{L}$ 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. $\bar{\mathcal{L}} = \mathcal{L}$.

From Corollary 1.5, we have that if E is a subspace of $\mathbb{R}^m$ of dimension r, then a basis $\mathbf{b}_1, \ldots, \mathbf{b}_r$ of the lattice $E \cap \mathbb{Z}^m$ can be extended to a basis $\mathbf{b}_1, \ldots, \mathbf{b}_m$ of $\mathbb{Z}^m$. This implies that a basis of $\bar{\mathcal{L}}$ can always be extended to a basis of $\mathbb{Z}^m$. Using this fact we can prove the following:

Lemma 1.8. *Let $\mathcal{L} \subseteq \mathbb{Z}^m$ be a lattice of dimension r. Then $\mathbb{Z}^m = \bar{\mathcal{L}} \oplus (\mathcal{L}^\perp)^\vee$.*

Proof. Let us consider a basis $\mathbf{b}_1, \ldots, \mathbf{b}_r$ of $\bar{\mathcal{L}}$. We can extend it to a basis $\mathbf{b}_1, \ldots, \mathbf{b}_m$ of $\mathbb{Z}^m$. Since $\mathbb{Z}^m = (\mathbb{Z}^m)^\vee$, its dual basis $\mathbf{v}_1, \ldots, \mathbf{v}_m$ is another basis of $\mathbb{Z}^m$, such that $\mathbf{B}\mathbf{V}^\intercal = \mathbf{I}_m$. In particular, $\mathbf{v}_{r+1}, \ldots, \mathbf{v}_m$ is a basis of $\mathcal{L}^\perp$, and $\mathbf{b}_{r+1}, \ldots, \mathbf{b}_m$ is a basis of $(\mathcal{L}^\perp)^\vee$.

□

The dimension of a lattice and its dual coincide. Thus, we obtain the following fact:

Corollary 1.9. *Let $\mathcal{L} \subseteq \mathbb{Z}^m$ be a lattice. Then* $\dim \mathcal{L} + \dim \mathcal{L}^\perp = m$.

Another important consequence of Lemma 1.8 is:

Corollary 1.10. *Let $\mathcal{L} \subseteq \mathbb{Z}^m$. Then* $\det(\mathcal{L}^\perp) = \det(\bar{\mathcal{L}})$.

Proof. We proved that $\mathbb{Z}^m = \bar{\mathcal{L}} \oplus (\mathcal{L}^\perp)^\vee$. Then $\det(\bar{\mathcal{L}}) \cdot \det(\mathcal{L}^\perp)^\vee = 1$ and therefore:

$$\det(\bar{\mathcal{L}}) \cdot \frac{1}{\det(\mathcal{L}^\perp)} = 1.$$

□

1.1.2 Lattice minima, Minkowski's theorem

In every lattice $\mathcal{L}$ we can find a non-zero vector $\mathbf{v}$ of minimal norm. The *first minimum* of $\mathcal{L}$ is $\lambda_1(\mathcal{L}) = \|\mathbf{v}\|$. Lattice points whose norm is the first minimum 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 [Min97] [Cas97, Section III.2.2], one can prove that for each $d \in \mathbb{N}^+$ $0 \leq \gamma_d \leq d/4 + 1$.

More generally, one can define:

Definition 1.11. *Let $\mathcal{L}$ be a lattice. For each $1 \leq i \leq \dim \mathcal{L}$, the* i-th minimum $\lambda_i(\mathcal{L})$ *is the minimum of* $\max_j \{\|\mathbf{v}_j\|\}$ *among all the sets $\{\mathbf{v}_j\}_{j \leq i}$ of i linearly independent lattice points.*

Theorem 1.12 (Minkowski's Second Theorem[1].). *Let $\mathcal{L}$ be a lattice of dimension d. For each $1 \leq i \leq d$*

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

Short vectors of lattices are interesting for many applications. For a "random lattice" it is known the first minimum by the *Gaussian Heuristic* [GN08, MW16, LN20]:

$$\frac{\lambda_1(\mathcal{L})}{\det(\mathcal{L})^{\frac{1}{d}}} \approx \sqrt{\frac{d}{2\pi e}}.$$

1.1.3 Reduced bases

Let $\mathbf{b}_1, \ldots, \mathbf{b}_d$ be an ordered basis of a lattice $\mathcal{L}$ of dimension d. Let $\mathbf{b}_1^*, \ldots, \mathbf{b}_d^*$ be the Gram-Schmidt orthogonalization, we define for $1 \leq j < i \leq d$

$$\mu_{i,j} = \frac{\langle \mathbf{b}_i, \mathbf{b}_j^* \rangle}{\langle \mathbf{b}_j^*, \mathbf{b}_j^* \rangle}.$$

Definition 1.13 (LLL reduced basis [LLL82]). *Given $1/4 < \delta < 1$, the basis $\mathbf{b}_1, \ldots, \mathbf{b}_d$ is* LLL reduced *(with factor δ) if the following properties hold:*

1. Size reduced: $|\mu_{i,j}| \leq 1/2$ *for* $1 \leq j < i \leq d$.
2. Lovász condition: $\|\mathbf{b}_j^*\|^2 \geq (\delta - \mu_{j,j-1}^2)\|\mathbf{b}_{j-1}^*\|^2$ *for each* $2 \leq j \leq d$.

Traditionally one takes $\delta = 3/4$ and LLL-reduced means reduced with respect to such a factor; in practice one takes $\delta = 0.99$. An LLL-reduced basis has many good properties.

Lemma 1.14. *Let $\mathbf{b}_1, \ldots, \mathbf{b}_d$ a LLL reduced basis of the lattice $\mathcal{L}$. Then*

1. $\|\mathbf{b}_1\| \leq 2^{\frac{d-1}{2}} \lambda_1(\mathcal{L})$;
2. $\|\mathbf{b}_j\| \leq 2^{\frac{d-1}{2}} \lambda_i(\mathcal{L})$ *for each* $1 \leq j \leq i \leq d$;
3. $2^{\frac{1-i}{2}} \lambda_i(\mathcal{L}) \leq \|\mathbf{b}_i\| \leq 2^{\frac{d-1}{2}} \lambda_i(\mathcal{L})$;
4. $\det(\mathcal{L}) = \prod_{j=1}^{d} \|\mathbf{b}_j^*\| \leq \prod_{j=1}^{d} \|\mathbf{b}_j\| \leq 2^{\frac{d(d-1)}{4}} \det(\mathcal{L})$.

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 the literature we can find other notions of reduced basis and algorithms [Sch87, GN08, MW16, ALNSD20]. For instance, one can consider the BKZ algorithm by Schnorr [Sch87],

[1] See [Ngu09]

which is the first block-wise reduction algorithm. Block-wise reduction algorithms can be seen as a generalisation the LLL algorithm. Informally, the quality of the returned reduced basis can be controlled by a parameter β, *i.e.* the block-size; namely, it holds the larger β given as input, the more refined the reduction. Let $\mathcal{L}(\mathbf{B})$ be any lattice of dimension d and $\mathbf{c}_1$ the first vector of the basis obtained as the output of a specific reduction algorithm. Then, we can usually define a property of the form

$$\|\mathbf{c}_1\| \leq \xi \ \cdot \det(\mathcal{L}(\mathbf{B}))^{\frac{1}{d}} \tag{1.2}$$

The values ξ and $\xi^{\frac{1}{n}}$ are called *Hermite factor* and *Hermite root factor* of the algorithm, respectively. They both can be used to quantify the quality of the reduction, mentioned above. Indeed, they provide an intuition on how close the algorithm is to be an SVP-oracle.

If a certain approximation of the shortest vector is needed, we can run BKZ-β with sufficiently large block size. Indeed, increasing β decreases its Hermite factor ξ_β. Notice that LLL corresponds to the lowest block-size $\beta = 2$. However, we have a trade-off between the quality of the approximation and the running time of the algorithm. Namely, Hanrot, Pujol and Stehlé showed in [HPS11] that some version of BKZ achieves $\xi_\beta = \beta^{\frac{d}{\beta}}$ by using a block-size β, but this requires heuristically at least $2^{\Omega(\beta)} \cdot \mathsf{poly}(d, \text{size}(\mathbf{B}))$ time. For instance, this implies that that we can get as Hermite factor $\sqrt{n}$ by using BKZ-β with $\beta \sim \omega(n)$, but in $2^{\Omega(n)}$ time. Recently, in [LN20] this result was extended to the original BKZ algorithm.

1.2 Computing the orthogonal lattice

To compute the orthogonal $\mathcal{L}^\perp$ of a lattice $\mathcal{L} \subseteq \mathbb{Z}^m$ of rank r one can apply the algorithm by Nguyen and Stern [NS97] based on LLL, which in practice is faster than standard kernel computation. Moreover, the output is an LLL-reduced basis of $\mathcal{L}^\perp$.

Letting $\mathbf{U}$ be a $r \times m$ basis matrix of row vectors of $\mathcal{L}$, the main idea is to consider the lattice

$$\mathcal{L}_c(\mathbf{U}) = \left[c \cdot \mathbf{U}^\intercal \mid \mathbf{I}_m\right]$$

where c is a positive integer constant. $\mathcal{L}_c(\mathbf{U})$ is a rank-m lattice of vectors in $\mathbb{Z}^{r+m}$. One first applies LLL to $\mathcal{L}_c(\mathbf{U})$ to obtain a basis $\mathbf{b}_1, \ldots, \mathbf{b}_m \in \mathbb{Z}^{r+m}$. Then one recovers a basis of $\mathcal{L}^\perp$ by projecting on the last m coordinates of the first $m - r$ vectors, *i.e.* $\pi(\mathbf{b}_1), \ldots, \pi(\mathbf{b}_{m-r})$ where $\pi(v_1, \ldots, v_{m+r}) = (v_{r+1}, \ldots, v_{m+r}) \in \mathbb{Z}^m$.

Lemma 1.15 ([CSV18]). *Let $\mathbf{U}$ be a basis of the lattice $\mathcal{L} \subseteq \mathbb{Z}^m$ of rank r. If $\mathbf{b}_1, \ldots, \mathbf{b}_m \in \mathbb{Z}^{r+m}$ is an LLL-reduced basis of $\mathcal{L}_c(\mathbf{U})$ with*

$$c > 2^{\frac{m-1}{2}} \cdot \lambda_{m-r}(\mathcal{L}^\perp), \tag{1.3}$$

then $\pi(\mathbf{b}_1), \ldots, \pi(\mathbf{b}_{m-r})$ is an LLL-reduced basis of $\mathcal{L}^\perp$.

Heuristic analysis

For a "random lattice" $\mathcal{L}$ we expect the minima of $\mathcal{L}^\perp$ to be balanced, and therefore $\lambda_{m-r}\left(\mathcal{L}^\perp\right)$ to be roughly equal to $\lambda_1\left(\mathcal{L}^\perp\right)$. This means that can we use the approximated bound:

$$\lambda_{m-r}\left(\mathcal{L}^\perp\right) \leq \sqrt{\gamma_{m-r}}\det(\mathcal{L}^\perp)^{\frac{1}{m-r}}$$

From Hadamard inequality, we obtain:

$$\lambda_{m-r}\left(\mathcal{L}^\perp\right) \leq \sqrt{\gamma_{m-r}}\det(\mathcal{L}^\perp)^{\frac{1}{m-r}} \leq \sqrt{\gamma_{m-r}}\det(\mathcal{L})^{\frac{1}{m-r}} \leq \sqrt{m}\cdot\|\mathbf{U}\|^{\frac{r}{m-r}}$$

where we denote by $\|\mathbf{U}\|$ the maximum Euclidean norm of the row vectors of $\mathbf{U}$. This means that for a "random lattice" it is sufficient to consider

$$c > 2^{\frac{m}{2}}\cdot\sqrt{m}\cdot\|\mathbf{U}\|^{\frac{r}{m-r}}. \tag{1.4}$$

For a rank-d lattice in $\mathbb{Z}^n$, the complexity of computing an LLL-reduced basis with the L^2 algorithm is $\mathcal{O}(d^4 n(d+\log B)\log B)$ without fast integer arithmetic, for vectors of Euclidean norm less than B. With $d=m$, $n=r+m$ and $B \leq c\|\mathbf{U}\| \leq 2^m\|\mathbf{U}\|^{m/(m-r)}$, the heuristic complexity of computing the orthogonal lattice of a rank-r lattice $\mathcal{L}(\mathbf{U}) \subseteq \mathbb{Z}^m$ is therefore:

$$\mathcal{O}\left(m^5\left(m+\frac{m}{m-r}\log\|\mathbf{U}\|\right)^2\right).$$

1.2.1 The orthogonal mod q

Let $\mathbf{a}\in\mathbb{Z}^m$ be any vector and q an integer, the *orthogonal to* $\mathbf{a}$ *mod* q is the lattice

$$\Lambda_q^\perp(\mathbf{a}) \coloneqq \{\mathbf{u}\in\mathbb{Z}^m \mid \langle\mathbf{u},\mathbf{a}\rangle \equiv 0 \pmod q\}.$$

This is a full-rank sublattice of $\mathbb{Z}^m$, since $q\mathbb{Z}^m \subseteq \Lambda_q^\perp(\mathbf{a})$. Therefore, it has finite index in $\mathbb{Z}^m$ and

$$\det(\Lambda_q^\perp(\mathbf{a})) = [\mathbb{Z}^m : \Lambda_q^\perp(\mathbf{a})] = \frac{q}{\gcd(q,a_1,\ldots,a_m)}.$$

Now, we explain how to construct a basis of $\Lambda_q^\perp(\mathbf{a})$. Since the case $\mathbf{a}=\mathbf{0} \mod q$ is trivial, we can assume without loss of generality that $a_1 \neq 0$. We write $\mathbf{u}=[u_1,\mathbf{u}']$ where $\mathbf{u}'\in\mathbb{Z}^{m-1}$. Similarly we write $\mathbf{a}=[a_1,\mathbf{a}']$ where $\mathbf{a}'\in\mathbb{Z}^{m-1}$. Now, suppose $\gcd(a_1,q)=1$. Since a_1 is invertible modulo q, we get:

$$\begin{aligned}\mathbf{u}\in\Lambda_q^\perp(\mathbf{a}) &\iff u_1\cdot a_1 + \langle\mathbf{u}',\mathbf{a}'\rangle \equiv 0 \pmod q\\ &\iff u_1 + \langle\mathbf{u}',\mathbf{a}'\rangle\cdot a_1^{-1} \equiv 0 \pmod q\end{aligned}$$

Therefore, a basis of $\Lambda_q^\perp(\mathbf{a})$ is given by the $m\times m$ matrix of row vectors:

$$\Lambda_q^\perp(\mathbf{a}) = \begin{bmatrix} q & \\ -\mathbf{a}' \cdot a_1^{-1}[q] & \mathbf{I}_{m-1} \end{bmatrix}$$

Notice that the assumption $\gcd(a_1, q) = 1$ implies $\det(\Lambda_q^\perp(\mathbf{a})) = q$. If such assumption is not satisfied, then $\gcd(q, a_1, \ldots, a_m) = d \neq 1$, and we can suppose that $\gcd(a_1/d, q) = 1$. Hence, similarly $a = a_1/d$ is invertible modulo q and

$$\Lambda_q^\perp(\mathbf{a}) = \begin{bmatrix} q/d & \\ -\mathbf{a}' \cdot a^{-1}[q] & \mathbf{I}_{m-1} \end{bmatrix}.$$

The number of positive integers coprime with q is counted by the Euler's totient function $\varphi(q)$. Then the probability that a number is invertible modulo q is $\varphi(q)/q$.

The multiplicativity of this function implies

$$\varphi(q) = q \prod_{\substack{p \mid q \\ p \text{ prime}}} (1 - 1/p)$$

Therefore, intuitively the value of $\varphi(q)/q$ strictly depends on the decomposability of q. For instance, if q is a pseudo prime this is very close to one, while if q is a power of 2 the probability is only $1/2$. In the following, we generally expect to be in the first case. Hence, Gaussian Heuristic suggests that in such a case almost always $\lambda_1(\Lambda_q^\perp(\mathbf{a})) \approx \sqrt{m} q^{1/m}$. Moreover, when q is prime we can prove a lower bound for the first lattice minimum:

Lemma 1.16. *Let q be a prime. Then with probability at least $1/2$ over $\boldsymbol{\alpha}$ sampled uniformly at random from $\mathbb{Z}_q^n$, we have $\lambda_1(\Lambda_q^\perp(\boldsymbol{\alpha})) \geq q^{1/n}/4$.*

Proof. We prove a lower bound in $q^{1/n}$ that applies for a significant fraction of the vectors $\boldsymbol{\alpha}$. More precisely, we prove using a counting argument that for a prime q, with probability at least $1/2$ over the choice of $\boldsymbol{\alpha}$, we have $\lambda_1(\Lambda_q^\perp(\boldsymbol{\alpha})) \geq q^{1/n}/4$. Given a non-zero vector $\mathbf{s}$ of dimension n with components strictly smaller than q, for prime q, there is a fraction $1/q$ of vectors $\boldsymbol{\alpha} \in \mathbb{Z}_q^n$, such that $\langle \mathbf{s}, \boldsymbol{\alpha} \rangle = 0 \pmod q$. Therefore each non-zero vector of norm $\beta < q$ can be a shortest vector of $\Lambda_q^\perp(\boldsymbol{\alpha})$ for a fraction at most $1/q$ of vectors $\boldsymbol{\alpha}$. Consider now all vectors $\mathbf{s}$ with $\|\mathbf{s}\| < \beta$. There are at most $(2\beta)^n$ such vectors, so they can be shortest vectors of a fraction at most $(2\beta)^n/q$ of vectors $\boldsymbol{\alpha}$. By taking β such that $(2\beta)^n/q < 1/2$, which gives $\beta < (q/2)^{1/n}/2$, we get that for a fraction at least $1/2$ of the choices of $\boldsymbol{\alpha}$, we have

$$\lambda_1\left(\Lambda_q^\perp(\boldsymbol{\alpha})\right) \geq (q/2)^{1/n}/2 \geq q^{1/n}/4.$$

□

1.3 Probability and statistics

Let x be a real-valued random variable. The behaviour of such kind of variables is fully described by the *cumulative distribution function* of x, which is usually denoted by $F_x(a) = \mathbb{P}(x \leq a)$. In the following, random variable always means real-valued random variable.

If the range of x is countable, x is called *discrete* and we can define the *probability mass function* as $p_x(a) = \mathbb{P}(x = a)$. While, if the range of x is continuous, x is called *continuous* and we can define the *probability density function* as a $p_x(a) = \frac{dF_x(x)}{dx}_{|x=a}$. This implies that $\mathbb{P}(x \leq a) = \int_{-\infty}^{a} p_x(\xi)d\xi$. Notice that the probability mass function can be seen as a probability density function with respect to the counting measures. Hence, in the following when x is discrete we always imply using such measure; namely, informally in the discrete case we interpret integrals as summations.

We define a *random vector* $\mathbf{x} = (x_1, \ldots, x_n)$ as a column vector whose coordinates x_i's are random variables. Similar to the univariate case we can define the cumulative distribution function as $F_{\mathbf{x}}(\mathbf{a}) = \mathbb{P}(\mathbf{x} \leq \mathbf{a})$. We say that the x_i's are *independent* if the cumulative distribution function of $\mathbf{x}$ is the product of the univariate cumulative distribution functions, *i.e.* $F_{\mathbf{x}}(\mathbf{a}) = \prod_{i=1}^{n} F_{x_i}(a_i)$.

Statistical distance for discrete distributions

Let X, Y be two random discrete variables having support included in the finite set S, their *statistical distance* is defined as

$$\Delta(X, Y) = \frac{1}{2} \sum_{n \in S} |p_X(n) - p_Y(n)|.$$

The statistical distance is a metric on distributions and has the following property [NSS01, Lemma 3]:

Lemma 1.17. *Let* $\mathbf{x} = (x_1, \ldots, x_m)$ *and* $\mathbf{y} = (y_1, \ldots, y_m)$ *two random vectors having both i.i.d. components distributed according to* X *and* Y*, respectively. Then:*

$$\Delta(\mathbf{x}, \mathbf{y}) \leq m\Delta(X, Y).$$

Expected value and moments

Let $g(x)$ a function of a random variable x. The *expected value* of $g(x)$ is

$$\mathbb{E}[g(x)] = \int_{\mathbb{R}} g(\xi)p_x(\xi)d\xi$$

This definition generalises directly to vectors as $\mathbb{E}[g(\mathbf{x})] = \int g(\boldsymbol{\xi})p_{\mathbf{x}}(\boldsymbol{\xi})d\boldsymbol{\xi}$. Using the properties of probability density functions and definitions, one can prove:

Lemma 1.18. *The expectation operator has the following properties:*

(Linearity 1) Given $\mathbf{y}_1, \dots, \mathbf{y}_k$ *random vectors and* $a_1, \dots, a_k$ *scalars, then* $\mathbb{E}[\sum_j a_j \mathbf{y}_j] = \sum_j a_j \mathbb{E}[\mathbf{y}_j]$.

(Linearity 2) Given $\mathbf{A} \in \mathbb{R}^{m \times n}$ *and* $\mathbf{x}$ *random vector of dimension* n*, then* $\mathbb{E}[\mathbf{A}\mathbf{x}] = \mathbf{A}\mathbb{E}[\mathbf{x}]$ *and* $\mathbb{E}[\mathbf{x}^\intercal] = \mathbb{E}[\mathbf{x}]^\intercal$.

(Composition) Given $\mathbf{x}, \mathbf{y}$ *random vectors. If there exists a vector function* $\mathbf{g}$ *such that* $\mathbf{y} = \mathbf{g}(\mathbf{x})$*, then* $\mathbb{E}[\mathbf{y}] = \mathbb{E}[\mathbf{g}(\mathbf{x})]$.

For $g(x) = x$, $\mathbb{E}[x]$ is the *mean*, or precisely the expectation of x, which is generally denoted by m_x. For a random vector, $\mathbb{E}[\mathbf{x}]$ is the vector whose components are the expectation of the components of $\mathbf{x}$ computed by using the marginal densities. In particular, we have $\mathbb{E}[\mathbf{x}]_i = m_{x_i}$. Thus, we define $\mathbf{m}_\mathbf{x} = \mathbb{E}[\mathbf{x}]$.

The *correlation* of two random variables x, y is $\mathbb{E}[xy]$. Given a vector $\mathbf{x}$, its *correlation matrix* $\mathbf{R}_\mathbf{x}$ is the matrix having as (i,j)th component the correlation of the ith and jth components of $\mathbf{x}$, *i.e.* $\mathbb{E}[x_i x_j]$. Therefore, $\mathbf{R}_\mathbf{x} = \mathbb{E}[\mathbf{x}\mathbf{x}^\intercal]$. This matrix is always symmetric and positive definite.

Two variables x, y are *uncorrelated* if their *covariance* $c_{xy} = \mathbb{E}[xy] - m_x m_y$ is zero. Notice that independent variables are uncorrelated, but the converse is false. Given a vector $\mathbf{x}$, its *covariance matrix* $\mathbf{C}_\mathbf{x}$ is the matrix of the covariance of the components of $\mathbf{x}$, namely $\mathbf{C}_\mathbf{x} = \mathbb{E}[(\mathbf{x} - \mathbf{m}_\mathbf{x})(\mathbf{x} - \mathbf{m}_\mathbf{x})^\intercal] = \mathbf{R}_\mathbf{x} - \mathbf{m}_\mathbf{x}\mathbf{m}_\mathbf{x}^\intercal$.

For $k \in \mathbb{N}$ we define the *k-th moment* of a random variable x as $\mu'_{k,x} = \mathbb{E}[x^k]$, and the *k-th central moment* as $\mu_{k,x} = \mathbb{E}[(x - m_x)^k]$. Obviously, $\mu'_{0,x} = \mu_{0,x} = 1$, $\mu'_{1,x} = m_x$ and $\mu_{1,x} = 0$. Moreover, the second central moment is usually called *variance* and denoted either by σ_x^2 and $\mathrm{Var}(x)$. The square root of the variance σ_x is referred as standard deviation. The variance is always non-negative, and when it is non-zero the *standardised central moments* $\tilde{\mu}_{k,x} = \mu_{k,x}/\sigma_x^k$ are well defined. In particular, notice that $\tilde{\mu}_{4,x} > 1$. A useful function of the moments is the *(excess) kurtosis*, *i.e.* $\kappa_x = \tilde{\mu}_{4,x} - 3$.

In the following, we usually consider random variables uniformly distributed over a set S, and we denote this distribution as $\mathcal{U}(S)$. Throughout the thesis, $s \leftarrow S$ indicates that the random variable s is uniformly distributed over a given set S. Moreover, in such a case we often use S instead of s as subscript, *e.g.* we denote by $\mu_{k,S}$ the central moments of s. If $J = \{0, \dots, B\}$, the mean of $\mathcal{U}(J)$ is $m_J = B/2$ and its central moments are $\mu_{0,J} = 1$, $\mu_{1,J} = 0$, $\mu_{2,J} = \sigma_J^2 = B(B+2)/2$, $\mu_{3,J} = 0$ and $\mu_{4,J} = \sigma_J^2(9\sigma_J^2 - 1)/5$.

Estimation of expected values

In practice the probability density function of a random vector may be unknown. Nevertheless, if a set of samples $\mathfrak{X} = \{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$ is available, we can compute *sample estimations* of expectations by the following formula

$$\hat{\mathbb{E}}[g(\mathbf{x})] \coloneqq \frac{1}{N} \sum_{i=1}^{N} g(\mathbf{x}_i).$$

Special cases which we are interested in are

$$\hat{\mathbf{m}}_{\mathbf{x}} = \hat{\mathbb{E}}[\mathbf{x}] = \frac{1}{N}\sum_{i=1}^{N}\mathbf{x}_i \quad \text{and} \quad \hat{\mathbf{C}}_{\mathbf{x}} = \hat{\mathbb{E}}[(\mathbf{x}-\hat{\mathbf{m}}_{\mathbf{x}})(\mathbf{x}-\hat{\mathbf{m}}_{\mathbf{x}})^{\intercal}] = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{x}_i-\hat{\mathbf{m}}_{\mathbf{x}})(\mathbf{x}_i-\hat{\mathbf{m}}_{\mathbf{x}})^{\intercal}$$

i.e. the *sample mean* and the *sample covariance matrix*, respectively.

1.4 The subset sum problem

When talking about *subset sum problem*, we generally refer to a particular problem belonging to a class of equivalence of problems.

Definition 1.19 (Decision subset sum problem). *Let n be an integer. Given $\alpha_1, \dots, \alpha_n \in \mathbb{Z}$ and $t \in \mathbb{Z}$, decide if there exist $x_1, \dots, x_n \in \{0,1\}$ such that*

$$t = \alpha_1 x_1 + \cdots + \alpha_n x_n.$$

Definition 1.20 (Computational subset sum problem). *Let n be an integer. Given $\alpha_1, \dots, \alpha_n \in \mathbb{Z}$ and $t \in \mathbb{Z}$, compute $x_1, \dots, x_n \in \{0,1\}$ such that*

$$t = \alpha_1 x_1 + \cdots + \alpha_n x_n$$

if they exist.

Informally, saying that these problems are equivalent means that we have a mutual polynomial reduction. Indeed, obviously DSSP reduces to CSSP; conversely, suppose that $\mathcal{A}$ is an algorithm for DSSP that outputs either 0 or 1 whether a solution exists or not, respectively, then $x_1 = \mathcal{A}((\alpha_2, \dots, \alpha_n), t)$ and $x_j = \mathcal{A}((x_1\alpha_1, \dots, x_{j-1}\alpha_{j-1}, \alpha_{j+1}, \dots, \alpha_n), t)$.

Furthermore, some applications require to consider modular sums; thus, we can also define modular variants of the forementioned problems. For instance, the computational problem becomes:

Definition 1.21 (Computational Modular subset sum problem). *Let n, Q be integers. Given $\alpha_1, \dots, \alpha_n \in \mathbb{Z}$ and $t \in \mathbb{Z}$ compute $x_1, \dots, x_n \in \{0,1\}$ such that*

$$t = \alpha_1 x_1 + \cdots + \alpha_n x_n \mod Q$$

if they exist.

As observed by Howgrave-Graham and Joux in [HGJ10], the computational and the computational modular problems are also equivalent. Obviously, if $\mathcal{A}(Q)$ is an algorithm for modular subset sum this can be applied with modulus $Q = \max(t, \sum \alpha_i) + 1$ to solve the regular CSSP. Conversely, if $\mathcal{B}$ is a solver for the subset sum, considering $t, \alpha_i \in [0, Q) \cap \mathbb{Z}$ we have that t must lie in $[0, n(Q-1)] \cap \mathbb{Z}$; this implies that it is sufficient to compute $\mathcal{B}(t + xQ)$ for $x = 0, \dots, n-1$.

The decision subset sum problem is known to be NP-complete [GJ90]; so, it has been used for constructing several cryptographic primitives. The first example is the Merkle-Hellman public key cryptosystems [MH78]. Later, other constructions were for example proposed by Impagliazzo and Naor [IN96], Chor and Rivest [CR88], Lyubashevsky *et al.* [LPS10] and Ajtai and Dwork [AD97]. In particular, the latter work links the hardness of the subset problem to the *shortest vector problem*, *i.e.* they show that breaking their scheme is as hard as solving instances of SVP.

Among the subset sum based systems, many are not really proven as secure as random instances of the standard subset sum problem, but rather they apply some transformation to subset sums. For instance, in the Merkle-Hellman cryptosystems superincreasing sequences are used for the decryption. For this reason several constructions have been broken during the years; see [Odl90]. Nevertheless, those systems that have been proven as secure as the subset sum problem remained unbroken for some choices of parameters.

Subfamilies of instances are generally characterised by the *subset sum density*, which expresses the proportion between the number n and the size of the *weights* $\alpha_1, \ldots, \alpha_n$. In the following, we mainly focus on the modular variant and we take as density the value

$$d = \frac{n}{\log Q}.$$

When $Q = \mathsf{poly}(n)$, we generally talk about *high-density* subset sums, and we can find many efficient algorithms to solve the problem in the literature, *e.g.* dynamic programming strategies [MST90]. Hence, this implies that using high-density subset sums is not convenient for cryptography. Despite, instead, *low-density* subset sums (*i.e.* $d < 1$) may seem convenient, since they statistically have a single solution and they are not substantially affected by those attacks, they revealed weaker in practice. Indeed, it was first proven by Lagarias-Odlyzko [LO85] that given access to an SVP-oracle, almost all subset sums with density about $d < 0.6463$ can be broken. Subsequently, this result was improved to $d < 0.9408$ by Coster, *et al.* [CJL+92]. Although theoretically this fact does not undermine the hardness of the subset sum problem, in practice these algorithms for small values of n have competitive performances since LLL and BKZ simulate quite well such oracles. In particular, in [LO85] is actually proven that for almost all subset sums having density $d = \mathcal{O}(1/n)$ the lattice based algorithm solves the problem in polynomial time. Finally, for $\log Q \approx n$ only exponential time algorithms are known. Namely, the best running time currently known is of the order of $2^{0.291n}$ on average; this algorithm by Becker, Coron and Joux [BCJ11] is an extension of the Howgrave-Graham and Joux [HGJ10] technique, having running time $O(2^{0.337n})$.

Further information regarding the distribution of subset sums will be recalled in Section 1.5; while we conclude this section by giving a simple intuition of the lattice based algorithm for solving the computational subset sum (Definition 1.20) by Lagarias and Odlyzko, as this somehow relates to algorithms discussed later in the corpus of the thesis. Given an instance of the weights $\alpha_1, \ldots, \alpha_n$ and a subset sum t, the idea is to construct a lattice $\mathcal{L}_U$ generated by the rows of the following $(n+1) \times (n+1)$ matrix, for a suitable U:

$$\begin{bmatrix} 1 & 0 & \cdots & 0 & -U\alpha_1 \\ 0 & 1 & \cdots & 0 & -U\alpha_2 \\ & & \ddots & & \vdots \\ 0 & & \cdots & 1 & -U\alpha_n \\ 0 & 0 & \cdots & 0 & Ut \end{bmatrix}$$

Now, if $\mathbf{e} = (x_1, \ldots, x_n) \in \{0,1\}^n$ is a solution, *i.e.* $t = \alpha_1 x_1 + \cdots + \alpha_n x_n$, we have that $\hat{\mathbf{e}} = (x_1, \ldots, x_n, 0) \in \mathcal{L}_U$. Since $\|\hat{\mathbf{e}}\| \leq \sqrt{n}$, $\hat{\mathbf{e}}$ is also a short vector. Analysing this lattice, it is possible to show that under certain circumstances $\pm\hat{\mathbf{e}}$ are the shortest vectors of $\mathcal{L}_U$. Therefore, in such a case given an SVP-oracle one can compute $\mathbf{e}$ and consequently solve the subset sum problem. See [LO85, Fri86, CJL+92].

1.5 Distribution of modular sums

Linear combinations of random integers are used to generate hard instances of lattice problems in lattice-based cryptography, since the fundamental work of Ajtai [Ajt96]. Several pieces of research consider random sums of randomly distributed integers, as for instance those mentioned in Section 1.4 and the generator that we will present in Section 1.6. Therefore, in this section we recall the results of Nguyen, Shparlinski and Stern [NSS01] about the distribution of modular sums and we discuss some consequences.

Let Q be a positive integer and $\mathbb{Z}_Q$ the group of integers modulo Q. For fixed n, given $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n) \in \mathbb{Z}_Q^n$ and a set $\mathcal{B} \subseteq \mathbb{Z}_Q^n$ a *modular sum* is

$$\langle \boldsymbol{\alpha}, \mathbf{x} \rangle \pmod{Q}$$

for some $\mathbf{x} \in \mathcal{B}$. Here, we are mainly interested in the following parametric families of sets: $\mathcal{B}_n(\Theta) = \{0, \ldots, \Theta\}^n$ and $\mathcal{B}_{n,\kappa}(\Theta)$, *i.e.* subset of $\{0, \ldots, \Theta\}^n$ of vectors having Hamming weight exactly κ. In the following, we omit Θ if equal to one.

For any given $c \in \mathbb{Z}_Q$, let us define $N_{\boldsymbol{\alpha}}(\mathcal{B}, c) := \{\mathbf{x} \in \mathcal{B} \colon \langle \boldsymbol{\alpha}, \mathbf{x} \rangle = c \pmod{Q}\}$ and $\mathbb{P}_{\boldsymbol{\alpha}}(\mathcal{B}, c) = N_{\boldsymbol{\alpha}}(\mathcal{B}, c)/|\mathcal{B}|$. Namely, $\mathbb{P}_{\boldsymbol{\alpha}}(\mathcal{B}, c)$ is the probability for a random vector sampled uniformly from $\mathcal{B}$ of being a solution of the congruence $c = \langle \boldsymbol{\alpha}, \mathbf{x} \rangle \pmod{Q}$. Nguyen *et al.* proved the following important theorem:

Theorem 1.22 ([NSS01, Theorem 2]). *For any $\mathcal{B} \subseteq \mathbb{Z}_Q^n$, the identity*

$$\frac{1}{Q^n} \sum_{\boldsymbol{\alpha} \in \mathbb{Z}_Q^n} \sum_{c \in \mathbb{Z}_Q} \left(\mathbb{P}_{\boldsymbol{\alpha}}(\mathcal{B}, c) - \frac{1}{Q} \right)^2 = \frac{Q-1}{Q|\mathcal{B}|} \tag{1.5}$$

holds.

This fact has a large number of corollaries and applications, *e.g.* see [HGJ10]. In particular, it implies that

$$\frac{1}{Q^n} \sum_{\boldsymbol{\alpha} \in \mathbb{Z}_Q^n} \sum_{c \in \mathbb{Z}_Q} \left| \mathbb{P}_{\boldsymbol{\alpha}}(\mathcal{B}, c) - \frac{1}{Q} \right| \leq \sqrt{\frac{Q}{|\mathcal{B}|}} \tag{1.6}$$

Depending on the context, the previous inequality may be interpreted differently. Let

$$\Delta_\alpha(\mathcal{B}) = \frac{1}{2} \sum_{c \in \mathbb{Z}_Q} \left| \mathbb{P}_{\boldsymbol{\alpha}}(\mathcal{B}, c) - \frac{1}{Q} \right|$$

be the statistical distance between uniform distribution over $\mathbb{Z}_Q$ and the modular sums distribution for fixed $\boldsymbol{\alpha}$ and Q. If $\delta = \mathbb{E}[\Delta_\alpha(\mathcal{B})]$ is the mean over the possible choices of $\boldsymbol{\alpha}$, Markov's inequality implies that for a given $c > 0$, the probability that $\Delta_\alpha(\mathcal{B}) < 2^c\delta$ is at least $1 - 2^{-c}$.

Now, suppose $\mathcal{B} = \mathcal{B}_n$, then $\mathcal{B}_n = 2^n$. Then, from (1.6) we can derive $\mathbb{E}[\Delta_\alpha(\mathcal{B}_n)] \leq \sqrt{Q}/2^{\frac{n}{2}+1}$ and we obtain that for any $c > 0$

$$\Delta_\alpha(\mathcal{B}_n) \leq \sqrt{Q}/2^{\frac{n}{2}-c+1} \tag{1.7}$$

with probability $1 - 2^{-c}$.

In particular, we have that for fixed Q and $\boldsymbol{\alpha}$ the distribution of subset sum outputs is exponentially close to uniform; namely, there exists $c' > 0$ such that for $\log Q \leq c'n$ the distribution of the modular (subset) sums is close to $\mathcal{U}(\mathbb{Z}_Q)$ with high probability. This is coherent with Section 1.4. In particular, we recall that Impagliazzo and Naor in [IN96] proposed a family of *universal one-way hash functions* and *pseudorandom generators* provably as hard as solving the subset sum problem for certain choices of parameters. These parameters are identified by an equivalent of (1.6), obtained from applying the Leftover Hash Lemma [HILL99].

Inequality (1.6) suggests that we can expect a similar behaviour when the modular sums are generated using $\mathcal{B}_{n,\kappa}$, for κ sufficiently large. Indeed, for instance we have that when $\kappa = n/2$ we can expect a random distribution, since $\mathcal{B}_{n,\kappa} = \binom{n}{\kappa}$ and asymptotically $\binom{n}{n/2} \sim 2^{n+1}/\sqrt{2\pi n}$. Conversely, if κ is a constant function of n we have that $\binom{n}{\kappa} \sim n^\kappa/\kappa!$, hence $\mathcal{B}_{n,\kappa} = \mathsf{poly}(n)$. This implies that, in this case, it is convenient to consider as the fundamental parameter $\lambda = \binom{n}{\kappa}$, rather than n, in order to evaluate the randomness.

1.6 BPV and EBPV generators of discrete log pairs

Boyko *et al.* presented in [BPV98] a generator of discrete logarithm pairs (x, g^x) that 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.

Basic BPV generator $\mathcal{G}_{n,\kappa}$. 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 Q.

Preprocessing Step: Take $\alpha_1, \ldots, \alpha_n \leftarrow \mathbb{Z}_Q$ 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 $d = \sum_{j \in S} \alpha_j \pmod Q$, if $d = 0$ restart, otherwise compute $D = \prod_{j \in S} \beta_j \pmod p$. Return (d, D).

The authors showed that if all the discrete logarithms of the previous outputs are unknown, computing the discrete log of any future output is at least as hard as solving an arbitrary discrete logarithm within the same setting.

Theorem 1.23 (Theorem 5 [BPV98]). *Let $\{(b_i, B_i)\}$ a set of output of $\mathcal{G}$ of length ℓ. If there exists an algorithm that given $\{B_i\}$ computes the discrete log of the next output of $\mathcal{G}$ with success rate ϵ, then there exists an algorithm computing the discrete log on any input in expected time $\mathcal{O}(1/\epsilon)$.*

However, in many applications of the generator in cryptographic protocols some of these values may get known. The authors claim that in such cases the security of the generator is related to the hardness of the *hidden subset sum problem*, see Definition I. In [NS99] where hardness of this problem was first discussed, Nguyen and Stern showed that depending on the security notion addressed, trying to break the BPV fast generator may also lead to an *affine* variant of the hidden subset sum problem, see Definition II. Indeed, they described a very nice passive attack against the generator used in Schnorr's signatures [Sch90], based on the affine hidden subset sum problem; the attack is also applicable to ElGamal [Elg85] 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:

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

Namely, consider 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 $Q = q$.

The hardness of hidden subset sum problem and its affine variant is extensively discussed in the following chapters.

Later in 2001, Nguyen *et al.* proposed an *extended BPV generator* (EBPV) [NSS01], including small exponentiations.

Extended BPV generator $\mathcal{G}_{n,\kappa,\theta}$. 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 Q.

Preprocessing Step: Take $\alpha_1, \ldots, \alpha_n \leftarrow \mathbb{Z}_Q$ 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 $|S| = \kappa$; for each $j \in S$ randomly generate $x_j \in \{1, \ldots, \theta - 1\}$ compute $d = \sum_{j \in S} \alpha_j \cdot x_j \pmod Q$, if $d = 0$ restart, otherwise compute $D = \prod_{j \in S} \beta_j^{x_j} \pmod p$. Return (d, D).

The authors also explain how to use the generator by including server communications to reduce the overall memory. Either the BPV and EBPV generators, both in their classic and revised form, have been used for applications, such as server delegation [NSS01, CLV21] wireless network systems [ABC+17, OY17], drones [MV22, ZJL+21, OY18, LYZ+20] and others [CMT01, PU19, KRS+12, KRS+11, ZAR17].

Regarding the security of the EBPV generator, the authors consider the implementation of the generator inside any signature scheme consisting of three algorithms: a key-generation $\mathcal{A}_k$, a signing $\mathcal{A}_s$ and a verification $\mathcal{A}_v$; where $\mathcal{A}_s$ calls a probabilistic random oracle returning DL pairs $(c, g^c \bmod p)$, then being replaced by the EBPV generator. They model an attacker that can make exactly m queries to the signature algorithm on messages of his choice (*adaptive attack*) and succeeds in *existential forgery* if it produces a valid signature for an unqueried message. Let T be such an attacker and T^* another attacker behaving according to this model applied to the scheme, with the same parameters, but where the random oracle is substituted by the EBPV generator. Then, the pair (T, T^*) can be considered in order to study the security of using of the generator. Indeed, the authors prove that if the distribution of its output is sufficiently close to uniform, the new signature scheme is secure if the original one is secure. Specifically, if $\mathbb{P}_{\mathbf{a}}(n, \kappa, \theta, m, T^*)$ and $\mathbb{P}_{\mathbf{a}}(m, T)$ are the probabilities of success of existential forgery for identical choices of keys, Theorem 4 of [NSS01] states that

$$\frac{1}{Q^n} \sum_{\mathbf{a}} |\mathbb{P}_{\mathbf{a}}(n, \kappa, \theta, m, T^*) - \mathbb{P}_{\mathbf{a}}(m, T)| \leq m \sqrt{Q / |\mathcal{B}_{n,\kappa}(\theta - 1)|} \tag{1.8}$$

where recall $\mathcal{B}_{n,\kappa}(\theta - 1)$ is the set of n dimensional vector with coefficients $\{0, \ldots, \theta - 1\}$ of Hamming weight κ. Their argument for the proof is actually very generic, as it relates the security to the distribution of modular sums. As we summarised in the previous section, they provide results regarding the distributions of the sum $\langle \mathbf{a}, \mathbf{x} \rangle \bmod Q$ for $\mathbf{x} \in \mathcal{B}$ for $\mathcal{B}$ any set. Therefore, the same inequality above can be obtained for different distributions. Obviously, modifying the underlying distribution changes the selection of suitable parameters.

Active and passive attacks described in [NS99] (similar to those mentioned earlier in this section) can be directly applied when the EBVP generator is used, too. However, the effectiveness of those attacks now does not rely on the hardness of the hidden subset sum problem, but

rather on the hardness of another problem, *i.e.* the *hidden linear combination problem* (Definition III). The hardness of this further variant of the hidden subset sum problem is discussed in Chapter 6.

2

Introduction to the hidden subset sum problem

The hidden subset sum problem is a variant of the classical subset sum problem where the n weights α_i are also hidden. Namely, let Q be an integer, and let $\alpha_1, \ldots, \alpha_n$ be integers in $\mathbb{Z}_Q$. Let $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{Z}^m$ be vectors with components in $\{0, 1\}$. Let $\mathbf{h} = (h_1, \ldots, h_m) \in \mathbb{Z}^m$ satisfying:

$$\mathbf{h} = \alpha_1 \mathbf{x}_1 + \alpha_2 \mathbf{x}_2 + \cdots + \alpha_n \mathbf{x}_n \pmod{Q} \tag{2.1}$$

The *hidden subset sum problem* consists in recovering 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, given the *modulus* Q and the *sample vector* $\mathbf{h}$ (Definition I).

The hardness of the hidden subset sum problem was first discussed by Nguyen and Stern in their paper [NS99] presented at Crypto'99, 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].

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 $\mathcal{O}(1/n)$. Additionally, provided a shortest vector oracle, the classical subset sum problem can be solved when the density d is less than $\simeq 0.94$, by the algorithm is based on finding a shortest vector in a lattice built from $h, \alpha_1, \ldots, \alpha_n, Q$ [CJL+92]. See Section 1.4. Although the vector $\mathbf{h}$ of a given hidden subset sum problem is a collection of subset sums, the aforementioned attack is clearly not applicable since the weights α_i's are hidden. Moreover, in practice the coordinates of $\mathbf{h}$ may be sampled from a prescribed distribution, as for example if associated to the BPV generator[1], and we will see that this has an impact of the hardness of the problem.

In order to properly study the hidden subset sum problem, with special regard to cryptography, we construct in this chapter a basic framework that will allow us to both provide more detailed explanations and present our contributions consistently in the next chapters.

[1] See Section 1.6.

2.1 "Random" HSSP

Cryptographic schemes are usually based on average-case hard problems. Informally, one usually shows that a protocol is secure, under a specific security notion, by proving a reduction to a problem which is supposed to be hard on *average*, *i.e.* such that a "random" instance of such a problem is difficult to solve. For example, most of the modern *lattice-based cryptography* is based on the two main average-case problems, SIS and LWE. These assumptions are considered robust since they have been proven to admit some *worst-case/average-case* reductions [Reg09, Pei09, MR07, Pei16]. In order to discuss the hardness of the hidden subset sum it is therefore convenient to specify the notion of "random" HSSP. This is indeed the scope of the current section.

Although the notion of random instance may be quite intuitive at a high level, a proper definition is necessary in order to construct a reliable framework. Hence, as a first task we have to identify the main parameters of the problem:

n is a positive integer, this is the number of weights $\alpha_1, \dots, \alpha_n$ and will be the main hardness parameter;
m is a positive integer, this is the dimension of the vectors $\mathbf{x}_1, \dots, \mathbf{x}_n, \mathbf{h}$;
Q is the modulus of the problem.

Notice that the hidden subset sum problem has an extra parameter compared to the traditional subset sum problem, *i.e.* the *sample size* m. Recall that a "random" instance of the latter problem is usually produced by sampling the weights $\alpha_1, \dots, \alpha_n$ uniformly random over $\mathbb{Z}_Q$ and computing $h = \sum_i \epsilon_i \alpha_i \pmod{Q}$ with ϵ_i variables uniformly random over $\{0, 1\}$ [LPS10]. Hence, proceeding coherently we obtain the following definition:

Definition 2.1 ($\mathrm{HSSP}_n(m, Q)$). *A* random instance *of* $\mathrm{HSSP}_n(m, Q)$ *is produced by sampling the weights* $\alpha_1, \dots, \alpha_n$ *uniformly at random over* $\mathbb{Z}_Q$ *and the vectors* $\mathbf{x}_1, \dots, \mathbf{x}_n \in \mathbb{Z}^m$ *with independent components sampled uniformly over* $\{0, 1\}$*, and computing*

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

according to Definition I.

The family $\mathrm{HSSP}_n(m, Q)$ is consistent when considering the self-standing problem, however the subset sums that naturally emerge from the BPV generator actually have an additional parameter. Recall that to generate a pair $(g, g^x \bmod p)$, the algorithm randomly generates a subset $S \subseteq [1, n]$ such that $|S| = \kappa$, then it computes $b = \sum_{j \in S} \alpha_j \pmod{Q}$. This is equivalent to sampling a vector $\boldsymbol{\epsilon}$ uniformly over $\mathcal{B}_{n,\kappa}$, *i.e.* the set of binary vectors of dimension n and Hamming weight κ, and computing $h = \langle \boldsymbol{\alpha}, \boldsymbol{\epsilon} \rangle \pmod{Q}$.

Differently from the previous case, here the components of the vector $\boldsymbol{\epsilon}$ are not independent. This implies the Definition 2.1 is not suitable for performing the cryptanalysis of the BPV generator. Indeed, in this case the vectors $\mathbf{x}_1, \dots, \mathbf{x}_n$ are not independently distributed. Hence, we define an additional family of hidden subset sum problems:

Definition 2.2 ($\mathrm{HSSP}_n^\kappa(m,Q)$**).** *A* random instance *of* $\mathrm{HSSP}_n^\kappa(m,Q)$ *is produced as follows:*

1. *the weights* $\boldsymbol{\alpha}=(\alpha_1,\ldots,\alpha_n)$ *are sampled uniformly at random over* $\mathbb{Z}_Q$*;*
2. $\mathbf{X}\in\mathbb{Z}^{n\times m}$ *is a binary matrix whose columns are vectors sampled independently and uniformly at random from* $\mathcal{B}_{n,\kappa}$*;*
3. $\mathbf{h}=\boldsymbol{\alpha}\cdot\mathbf{X} \pmod Q$*;*
4. *the vectors* $\mathbf{x}_1,\ldots,\mathbf{x}_n$ *are the rows of* $\mathbf{X}$*.*

Generally, it is convenient to take into account both families of problems when discussing the *average-hardness* of the hidden subset sum problem, and we will show that some *average-properties* can be exploited differently.

It is useful to define further features in order to identify subfamilies of problems. Coherently with [NS99], we define the *density* as $d=n/\log Q$, which is essentially the density of the subset sums. This actually differs from the definition of [LO85], while it is closer to the framework established by Howgrave-Graham and Joux, in which this is referred as prescribed density; see Section 2.2 of [HGJ10]. The density measures the proportion between the main parameter n and the bitsize of the modulus, which is also the bitsize of the coefficients of $\boldsymbol{\alpha}$ and $\mathbf{h}$. We remark that the given definition of density does not take into consideration the value of m. Although this may seem inconsistent with the setting of the hidden subset sum problem, our work will show that this is indeed a correct measure. See Remark 4.3.

We define the *sparsity* ς of a hidden subset sum problem as the (matrix) density of $\mathbf{X}$ the matrix whose rows are the vectors $\mathbf{x}_1,\ldots,\mathbf{x}_n$, *i.e.* the ratio between the number of non zero coordinates of $\mathbf{X}$ and nm. The expectation of the sparsity of a random instance of $\mathrm{HSSP}_n(m,Q)$ is $1/2$. Instead, a random instance of $\mathrm{HSSP}_n^\kappa(m,Q)$ has always fixed sparsity $\varsigma=\kappa/n$. We can characterise an hidden subset sum depending on its sparsity as *balanced* if $\varsigma\approx 1/2$, and *unbalanced*, otherwise; specifically, we call it *sparse* if $\varsigma\ll 1/2$.

We establish as a convention that by writing HSSP we refer to any one of the previous families; namely, we will specify the distribution only if the sampling procedure impacts either the discussion or the result we are analysing. Moreover, we omit the parameters m,Q when implicit.

About density and hardness of HSSP

In Section 1.4 we recalled that if $Q=\mathsf{poly}(n)$ we generally talk about *high-density* subset sums, while about *low-density* subset sums if $\log Q\gg n$. We also discussed the connection between density and known attacks. Then, in Section 1.5 we reviewed some facts about the distributions of modular sums. Specifically, we observed that if $d>1$ the distribution of the subset sums is close to uniform.

A similar result for $\mathrm{HSSP}_n(m,Q)$ is straightforward due to the composability of the statistical distance. Indeed, by using (1.7) and Lemma 1.17 we obtain that for $c>0$, the statistical distance between the distribution of any $\mathbf{h}$ generated as $\mathrm{HSSP}_n(m,Q)$ and a random vector uniformly sampled over $\mathbb{Z}_Q^m$ is lower than $m\sqrt{Q}/2^{\frac{n}{2}-c+1}$, with probability $1-2^{-c}$. Therefore,

when m is polynomial in n, we can expect that if $d = n/\log Q \gtrsim 1$, the sample vector $\mathbf{h}$ will be essentially distributed uniformly over $\mathbb{Z}_Q^m$. Namely, we have that we can expect high-density hidden subset sums vectors being indistinguishable from vectors sampled uniformly at random.

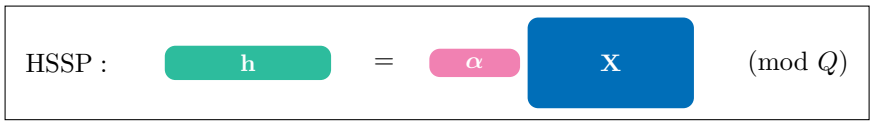
Similarly, in the model established by Definition 2.2, the coordinates of $\mathbf{h}$ are generated independently, hence Lemma 1.17 can still be applied. Let $\lambda = \binom{n}{\kappa}$, for simplicity we assume now $m = \mathsf{poly}(\log \lambda)$. The statistical distance between the distribution of a $\mathbf{h}$, in a random instance of $\mathrm{HSSP}_n^k(m, Q)$, and the uniform distributions over $\mathbb{Z}_Q^m$ is $mC\sqrt{Q/\lambda}$ with probability greater than $1 - 1/C$ for $C > 0$. This implies that sample vectors of balanced high-density $\mathrm{HSSP}_n^\kappa(m, Q)$, *i.e.* when $\varsigma \approx 1/2$ and $d \approx 1$, are indistinguishable from uniformly sampled vectors. For sparse instances, fewer samples are sufficient to get collisions using a modular sum generator, see Section 1.5. Hence, if λ is small, a guessing strategy might be efficient. Nevertheless, the distribution is close to uniform, if λ is large enough, *e.g.* $\log \lambda \approx n$ when $d > 1$. Therefore, for cryptographic applications whose security is related to the hardness of HSSP, density-one instances are a valid choice. Conversely, choosing an instance with either high-density or $\log \lambda = \mathcal{O}(1)$ may be not wise, since in the former case the problem can have many solutions, while in the latter the space of possible solutions is quickly exhaustible. Moreover, our work will show that low-density HSSP are not recommended, too; indeed, in such a case we can likely break the problem in polynomial time. Specifically, we derive a rigorous lower bound for the bitsize of the modulus Q implying that $d = \mathcal{O}(1/(n \log n))$, and heuristically $\mathcal{O}(1/n)$, is not advisable for cryptography.

2.2 Solving HSSP in two steps

In their paper [NS99] Nguyen and Stern proposed to divide the algorithm for solving the problem in two parts: a first one consisting of an algorithm for disclosing an hidden lattice containing the binary vectors, and a second one that is essentially a binary vectors' search inside such a lattice. We will show that the bottleneck of their algorithm resides in the second step, as they utilise a lattice reduction strategy for retrieving very short vectors. Indeed, for large values of n it results impractical.

Albeit this issue, this overall strategy is a considerable solution and a fruitful basis for the successive algorithms. Our new algorithms proceed according to the same model, *i.e.* in two steps. This first step is an improvement of the original Nguyen-Stern orthogonal lattice attack for disclosing the hidden lattice. For the second step, we totally replace their binary vectors' search, via lattice reduction, with two novel methods. Our new algorithms for the second step follow two different principles:

- The *multivariate method* reduces the problem of revealing the binary vectors to solving a multivariate quadratic system. We show that if $m \simeq n^2$, this method heuristically solves the problem in polynomial time. This attack is partially included in [CG20a] and described in Chapter 4.



Input: $\mathbf{h}$, Q
Output: $\mathbf{X}, \boldsymbol{\alpha}$
Algorithm:

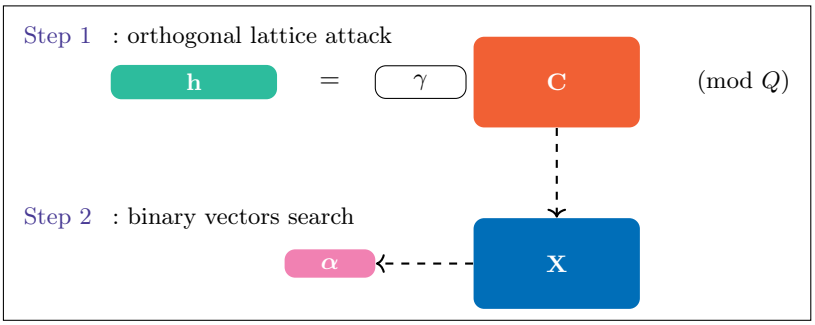


Fig. 2.1. Solving HSSP in two steps.

- The *statistical learning method* reduces the problem of revealing the binary vectors associated to HSSP_n to solving an instance of the *hidden parallelepiped problem* introduced by Nguyen and Regev in [NR09]. We propose two versions: a proven and a heuristic algorithm. In fact, since in order to *prove* that the algorithm is successful the input sample's size m required is very large, we also show that it possible to adapt this technique for heuristically solving HSSP_n with fewer samples, in only a few seconds. Furthermore, this second method can be adjusted for solving the *hidden linear combination problem*, too. The proven algorithm is presented in Chapter 5 and the heuristic one in Chapter 6. We presented a preliminary version of our statistical method in [CG22].

3

The Nguyen-Stern algorithm

Nguyen and Stern presented the first lattice based algorithm for solving the hidden subset sum problem [NS99] at Crypto'99. Their attack relies on the technique of the orthogonal lattice. This technique was introduced by Nguyen and Stern at Crypto '97 for the cryptanalysis of the Qu-Vanstone cryptosystem [NS97], and it was later applied for studying the security of several applications: as for example, the cryptanalysis of the Ajtai-Dwork cryptosystem [NS98b], of the Béguin-Quisquater server-aided RSA protocol [NS98a], and fault attacks against RSA-CRT signatures [CNT10, BNNT11], attacks against discrete-log based signature schemes [NSS04]. It has been also used for the cryptanalysis of numerous homomorphic encryption schemes [vDGHV10, LT15, FLLT15] and multilinear maps [CLT13, CP19, CN19].

The *orthogonal lattice attack* for the hidden subset sum problem is based on the following observation: if a vector $\mathbf{u}$ is orthogonal modulo Q to the public vector $\mathbf{h}$, then from (I) 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.

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. See Section 3.3.

3.1 The algorithm

The Nguyen-Stern algorithm for solving the hidden subset sum problem was first presented in [NS99]. In this section, we present the algorithm, and later in Section 3.2 we will provide our analysis.

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

$$\mathbf{h} = \alpha_1\mathbf{x}_1 + \alpha_2\mathbf{x}_2 + \cdots + \alpha_n\mathbf{x}_n \pmod{Q} \tag{3.1}$$

we have to 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 proceeds in 2 steps:

1. From the sample vector $\mathbf{h}$ and the modulus Q, 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 Q, recover the weights α_i.

Nguyen and Stern implemented the first step as an *orthogonal lattice attack*. Namely, they exploit the properties of the vectors orthogonal to $\mathbf{h}$ and the $\mathbf{x}_i$'s in order to construct a lattice containing $\mathcal{L}_{\mathbf{x}}$. For the second step they perform a lattice reduction for retrieving the hidden binary vectors.

3.1.1 First step: orthogonal lattice attack

The goal of the first step of the Nguyen-Stern algorithm is to recover the hidden lattice $\bar{\mathcal{L}}_{\mathbf{x}} \subseteq \mathbb{Z}^m$, *i.e.* the completion of the lattice $\mathcal{L}_{\mathbf{x}}$ generated by the $\mathbf{x}_i$'s, given the sample vector $\mathbf{h}$ and the modulus Q.

Let $\mathcal{L}_0$ be the lattice of vectors orthogonal to $\mathbf{h}$ modulo Q:

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

The main observation from [NS99] is that 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 Q, because from (3.1) we have

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

Hence, if $\mathbf{p_u}$ is shorter than the shortest non-zero vector orthogonal to $\boldsymbol{\alpha}$ modulo Q, the condition $\mathbf{p_u} = 0$ must be satisfied. This implies that in such a case $\mathbf{u} \in \mathcal{L}_{\mathbf{x}}^{\perp}$.

For binary the $\mathbf{x}_i$'s Nguyen and Stern observed that if the vector $\mathbf{u}$ is short, then $\mathbf{p_u}$ will be short, too. 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 $\bar{\mathcal{L}}_{\mathbf{x}}$.

The orthogonal lattice attack is summarised by Algorithm 3.1. While, methods for computing both $\mathcal{L}_0$ and $\mathcal{L}_{\mathbf{x}}^{\perp}$ are discussed in Section 1.2

Algorithm 3.1 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** $(\mathbf{c}_1, \ldots, \mathbf{c}_n)$

3.1.2 Second step: BKZ

The output of the first step, *i.e.* the orthogonal lattice attack, is an LLL-reduced basis $(\mathbf{c}_1, \ldots, \mathbf{c}_n)$ of the complete lattice $\bar{\mathcal{L}}_{\mathbf{x}} \subset \mathbb{Z}^m$. However, this is not necessarily composed by 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 $\mathbf{x}_i$, we have to apply BKZ instead of LLL. Indeed, a better approximation factor is obtained when using BKZ. Eventually from $\mathbf{h}$, the $\mathbf{x}_i$'s and Q, 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 Section 3.2.2, we will provide a heuristic analysis of the second step of the Nguyen-Stern algorithm, based on a model exploiting the gap between the shortest vectors of $\mathcal{L}_{\mathbf{x}}$ (the vectors $\mathbf{x}_i$'s), and the "generic" short vectors of $\mathcal{L}_{\mathbf{x}}$.

Recovering the vectors $\mathbf{x}_i$

Computing the binary vectors $\mathbf{x}_i$ we have to take into account that they are not the only short vectors in $\mathcal{L}_{\mathbf{x}}$; indeed, the vectors $\mathbf{x}_i - \mathbf{x}_j$ are roughly equally short, as later shown in Section 3.2.2.

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 $\mathcal{L}'_{\mathbf{x}} = 2\mathcal{L}_{\mathbf{x}} + \mathbf{1}\mathbb{Z}$, where $\mathbf{1} = (1, \ldots, 1)$. 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 $2\mathbf{x} - \mathbf{1} \in \mathcal{L}'_{\mathbf{x}}$ with components in $\{-1, 1\}$, hence shorter. This should enable to recover the secret vectors $\mathbf{x}_i$ as the shortest vectors in $\mathcal{L}'_{\mathbf{x}}$. We will see that the vector $\mathbf{1}$ plays a special role in HSSP_n^κ, and using this method may not be actually favourable.

A simpler method

We now describe a slightly simpler heuristic approach, in which we stay in the lattice $\bar{\mathcal{L}}_{\mathbf{x}}$, which is convenient especially for balanced HSSP.

However, we need to explain first the reason why for large enough values of m we are unlikely to obtain vectors in $\{0, \pm 1\} \cap \mathcal{L}_{\mathbf{x}}$ as combination of more that two $\mathbf{x}_i$'s if $\varsigma \approx 1/2$. 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 $m = \max(2n, 16 \log n + 24)$.

Now, suppose that the $\mathbf{x}_i$'s are the only binary vectors in $\bar{\mathcal{L}}_{\mathbf{x}}$. Hence, after BKZ reduction with a large enough block-size β as above, we expect that each one 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 $i \neq j$. To find the binary vector we can first recover the vectors $\mathbf{c}_i = \pm\mathbf{x}_j$ from the basis, and store them in a list L. Then, we look for new binary vectors of the form $\mathbf{c}_j \pm \mathbf{v}$ for $\mathbf{v} \in L$, updating the list L, and we iterate until no more vectors $\mathbf{v}$ have been added to the list L.

Algorithm 3.2 fully describes this strategy; we denote by `Check` the function taking as input a vector $\mathbf{v}$, and returning $\mathbf{v}$ or $-\mathbf{v}$ if $\mathbf{v} \in \{0, 1\}^m$ or $\mathbf{v} \in \{0, -1\}^m$ respectively; otherwise, it returns None.

Algorithm 3.2 Recovering the $\mathbf{x}_i$'s

Input: $\mathbf{c}_1, \ldots, \mathbf{c}_n, n, m$.
Output: $\mathbf{x_1}, \ldots, \mathbf{x_n}$.

```
L ← [ ]
for j ← 1 to n do
    Append Check(c_j) to L.
end for
for all v ∈ L do
    for j ← 1 to n do
        c ← Check(c_j − v)
        if c and c ∉ L then
            Append c to L.
        end if
        c ← Check(c_j + v)
        if c and c ∉ L then
            Append c to L.
        end if
    end for
end for
return L
```

After the first loop L must contain at least one of the $\pm\mathbf{x}_j$'s. Indeed, at least one of the $\mathbf{c}_i$ is one the target vectors; otherwise, the vector of all ones would belong to the kernel of the base change between $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ and $(\mathbf{c}_1, \ldots, \mathbf{c}_n)$. Then, suppose $L = \{\mathbf{c}_1, \ldots, \mathbf{c}_t\}$. The base change matrix is of the form

$$\mathbf{M} = \left(\begin{array}{c|c} \mathbf{I}_t & \mathbf{0} \\ \hline * & \mathbf{M}_{n-t} \end{array}\right)$$

Until $t < n$, for $i = t+1, \ldots, n$ and for any $\mathbf{v}$ in L the second loop computes $\mathbf{c} = \pm\mathbf{c}_i - \mathbf{v}$, and checks if either $\mathbf{c}$ or $-\mathbf{c}$ are suitable to be added to the set L. Applying the same argument as before to $\mathbf{M}_{n-t}$, we must find at least a new vector. The number of checks by round is $\mathcal{O}(n - \#L)$. Therefore, $\mathcal{O}(n^3)$ tests are performed.

We will show in our analysis that for HSSP_n^κ, it is not true that the $\mathbf{x}_i$'s are the only binary vectors in $\bar{\mathcal{L}}_{\mathbf{x}}$. Nevertheless, we will explain why this method can still be applied with a few modifications.

Recovering the weights $\boldsymbol{\alpha_i}$

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

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

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 $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n) \pmod{Q}$. Assuming that $\mathbf{X}'$ is invertible modulo Q, we get $\boldsymbol{\alpha} = \mathbf{X'}^{-1}\mathbf{h}' \pmod{Q}$.

3.2 Our analysis of Nguyen-Step algorithm

In this section we analyse the Nguyen-Stern algorithm, illustrated in Section 3.1.

The authors of [NS99] presented only a heuristic analysis of the orthogonal lattice attack. Thus, here we expand and adapt their arguments, obtaining a rigorous analysis. More precisely, we are able to prove that for a large enough modulus Q, 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. From this rigorous condition, we also derive further heuristics.

While for the second step, we provide a heuristic analysis, absent in the original paper. In spite of the efficient practical behaviour of Nguyen-Stern attack for small values of n, we argue that the actual running time of the second step is asymptotically exponential. In fact, we provide a simple model for the minimal BKZ block-size β that can recover the secret binary vectors, 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.

A preliminary version of this chapter has been published in the proceedings of Crypto2020 [CG20a].

3.2.1 First Step

Rigorous analysis

We provide a rigorous analysis of the orthogonal lattice attack described in Section 3.1.1. Specifically, we prove the following fact:

Theorem 3.1. *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 3.1 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 is $d = n/\log Q = \mathcal{O}(1/(n \log n))$.*

The proof is based on two lemmas: Lemma 3.2 states sufficient conditions for the correctness of Algorithm 3.1, depending on lattice minima; while, Lemma 1.16 determines an average lowers bound for $\Lambda_Q^{\perp}(\boldsymbol{\alpha})$. Recall that $\Lambda_Q^{\perp}(\boldsymbol{\alpha})$ denotes the lattice of vectors orthogonal to $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$ modulo Q, and that all the lattices considered here are integer lattices.

Lemma 3.2. *Assume that the lattice $\mathcal{L}_{\mathbf{x}}$ has rank n. Algorithm 3.1 computes a basis of the lattice $\bar{\mathcal{L}}_{\mathbf{x}}$ in polynomial time under the condition $m > n$ and*

$$\sqrt{mn} \cdot 2^{\frac{m}{2}} \cdot \lambda_{m-n}\left(\mathcal{L}_{\mathbf{x}}^{\perp}\right) < \lambda_1\left(\Lambda_Q^{\perp}(\boldsymbol{\alpha})\right). \tag{3.2}$$

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

$$\sqrt{mn}\|\mathbf{u}\| < \lambda_1\left(\Lambda_Q^{\perp}(\boldsymbol{\alpha})\right). \tag{3.3}$$

The lattice $\mathcal{L}_0$ is full rank of dimension m since it contains $Q\mathbb{Z}^m$. Now, consider $\mathbf{u}_1, \ldots, \mathbf{u}_m$ an LLL-reduced basis of $\mathcal{L}_0$. From Lemma 1.14, for each $j \leq m - n$ we have

$$\|\mathbf{u}_j\| \leq 2^{\frac{m}{2}} \cdot \lambda_{m-n}(\mathcal{L}_0) \leq 2^{\frac{m}{2}} \cdot \lambda_{m-n}\left(\mathcal{L}_{\mathbf{x}}^{\perp}\right) \tag{3.4}$$

since $\mathcal{L}_{\mathbf{x}}^{\perp}$ is a sublattice of $\mathcal{L}_0$ of dimension $m - n$. Combining with (3.3), this implies that when

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

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 3.1 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.

□

We can finally prove Theorem 3.1.

Proof (of Theorem 3.1). In order to apply Lemma 3.2, 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

$$\lambda_{m-n}\left(\mathcal{L}_{\mathbf{x}}^{\perp}\right) \leq \sqrt{\gamma_{m-n}}^{m-n} \det\left(\mathcal{L}_{\mathbf{x}}^{\perp}\right) \leq m^{m/2} \det\left(\mathcal{L}_{\mathbf{x}}^{\perp}\right). \tag{3.5}$$

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:

$$\det \mathcal{L}_{\mathbf{x}}^{\perp} \leq \det \mathcal{L}_{\mathbf{x}} \leq \prod_{i=1}^{n} \|\mathbf{x}_i\| \leq m^{n/2} \tag{3.6}$$

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 3.2, we can recover a basis of $\bar{\mathcal{L}}_{\mathbf{x}}$ when

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

From Lemma 1.16, 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:

$$\sqrt{mn} \cdot 2^{\frac{m}{2}} \cdot m^m < Q^{1/n}/4. \tag{3.7}$$

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

□

Improved bound for $\mathbf{HSSP}_{\boldsymbol{n}}^{\boldsymbol{\kappa}}$

Let us consider a random hidden subset sum problem according to Definition 2.2. Recall that $\mathbf{X} \in \{0,1\}^{n \times m}$ is the matrix whose rows are the vectors $\mathbf{x}_i$'s, then $\mathbf{h} = \boldsymbol{\alpha} \cdot \mathbf{X} \pmod{Q}$ and the columns of $\mathbf{X}$ have Hamming weight κ.

Denoting by $\mathbf{1}_s$ the vector of dimension s having all components equal to 1, the condition on the Hamming weight implies that

$$\mathbf{1}_n \cdot \mathbf{X} = \kappa \cdot \mathbf{1}_m \tag{3.8}$$

Recall that the completion of the lattice $\mathcal{L}_{\mathbf{x}}$ is $\bar{\mathcal{L}}_{\mathbf{x}} = E_{\mathcal{L}_{\mathbf{x}}} \cap \mathbb{Z}^m = (\mathcal{L}_{\mathbf{x}}^{\perp})^{\perp}$. Therefore, we have that $\mathbf{1}_m \in \bar{\mathcal{L}}_{\mathbf{x}}$ and consequently $\mathbf{y}_i = \mathbf{1}_m - \mathbf{x}_i \in \bar{\mathcal{L}}_{\mathbf{x}}$ for any $i = 1, \ldots, n$. If $\mathbf{1}_m \notin \mathcal{L}_{\mathbf{x}}$, this implies that the lattice is not complete. For the sake of simplicity, in the following we will always suppose that $\mathbf{1}_m \neq \mathbf{x}_i$ for each i.

Moreover, consider $\mathcal{L}_1$ the lattice generated by the matrix $\mathbf{X}'$ whose rows are $\mathbf{1}_m, \mathbf{x}_2, \ldots, \mathbf{x}_n$. We have that

$$\mathbf{X} = \begin{bmatrix} \kappa & -\mathbf{1}_{n-1} \\ \mathbf{0} & \mathbf{I}_{n-1} \end{bmatrix} \cdot \mathbf{X}'$$

and so $\det(\mathcal{L}_{\mathbf{x}}) = \kappa \det(\mathcal{L}_1)$. This implies that $\mathcal{L}_{\mathbf{x}} \subsetneq \mathcal{L}_1 \subseteq \bar{\mathcal{L}}_{\mathbf{x}}$, then $[\bar{\mathcal{L}}_{\mathbf{x}} \colon \mathcal{L}_{\mathbf{x}}] = [\bar{\mathcal{L}}_{\mathbf{x}} \colon \mathcal{L}_1][\mathcal{L}_1 \colon \mathcal{L}_{\mathbf{x}}]$. Therefore, $\kappa \det(\bar{\mathcal{L}}_{\mathbf{x}}) | \det(\mathcal{L}_{\mathbf{x}})$.

For equation (3.6) in the proof of Theorem 3.1 we used the relation $\det \mathcal{L}_{\mathbf{x}}^{\perp} = \det \bar{\mathcal{L}}_{\mathbf{x}} \leq \det \mathcal{L}_{\mathbf{x}}$. However, we just observed that for HSSP_n^{κ} this bound can be improved to $\det \mathcal{L}_{\mathbf{x}}^{\perp} = \det \bar{\mathcal{L}}_{\mathbf{x}} \leq \frac{1}{\kappa} \det \mathcal{L}_{\mathbf{x}}$. Thus, Equation (3.6) becomes

$$\det \mathcal{L}_{\mathbf{x}}^{\perp} \leq \frac{1}{\kappa} \det \mathcal{L}_{\mathbf{x}} \leq \frac{1}{\kappa} \prod_{i=1}^{n} \|\mathbf{x}_i\| \leq \frac{1}{\kappa} m^{n/2} \tag{3.9}$$

and consequently the sufficient condition is

$$\sqrt{mn} \cdot 2^{\frac{m}{2}} \cdot m^m / \kappa < Q^{1/n}/4. \tag{3.10}$$

Therefore, for HSSP_n^{κ} Theorem 3.1 can be slightly improved:

Theorem 3.3. *Let $m > n$. Assume that the lattice $\mathcal{L}_{\mathbf{x}}$ has rank n and the problem is a random instance of* HSSP_n^κ. *With probability at least* $1/2$ *over the choice of* $\boldsymbol{\alpha}$, *Algorithm 3.1 recovers a basis of $\bar{\mathcal{L}}_{\mathbf{x}}$ in polynomial time, assuming that Q is a prime integer of bitsize at least* $2mn \log m - n \log(\kappa)$.

Heuristic analysis for $\mathbf{HSSP}_n$

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 Q, namely we can take $\log Q = \mathcal{O}(n^2 \log n)$ when $m = 2n$. Now, we show that heuristically we can take $\log Q = \mathcal{O}(n^2)$, which gives a knapsack density $d = n/\log Q = \mathcal{O}(1/n)$. We also provide the concrete bitsize of Q used in our experiments, and a heuristic complexity analysis.

Heuristic size of the modulus Q

In order to derive a heuristic size for the modulus Q, we use an approximation of the terms in the condition (3.2) from Lemma 3.2.

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 (3.5) from the proof of Theorem 3.1, 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 (3.6), this gives:

$$\lambda_{m-n}\left(\mathcal{L}_{\mathbf{x}}^\perp\right) \lessapprox \sqrt{\gamma_{m-n}} m^{\frac{n}{2(m-n)}}. \tag{3.11}$$

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 (3.2) 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 (3.2) we obtain the heuristic condition:

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

This gives the condition:

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

which gives:

$$\log Q > \iota \cdot m \cdot n + \frac{n}{2}\log(n \cdot \gamma_{m-n}/\gamma_n) + \frac{mn}{2(m-n)}\log m. \tag{3.12}$$

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

$$\log Q > 2\iota \cdot n^2 + \frac{3n}{2} \log n + n \tag{3.13}$$

which gives $\log Q = \mathcal{O}(n^2)$ and a knapsack density $d = n/\log Q = \mathcal{O}(1/n)$. In practice for our experiments we use $m = 2n$ and $\log Q \simeq 2\iota n^2 + n \log n$ with $\iota = 0.035$. Finally, we note that smaller values of Q 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 $\mathcal{O}(m^5(m + \log Q) \log Q)$. 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 (3.4) and (3.11), we can heuristically assume $\|\mathbf{U}\| \leq 2^{m/2} \cdot \sqrt{m} \cdot m^{\frac{n}{2(m-n)}}$. 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 Section 1.2. 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 summarise the complexities of the two LLL operations in Table 3.2.1; 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 $d = n/\log Q = \mathcal{O}(1/n)$, as in the classical subset sum problem.

m	$\log Q$	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	$\mathcal{O}(n^3)$	$\mathcal{O}(n^{16})$	$\mathcal{O}(n^{16})$
$2n$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^9)$	$\mathcal{O}(n^7)$
$n + 1$	$\mathcal{O}(n^2 \log n)$	$\mathcal{O}(n^9 \log^2 n)$	$\mathcal{O}(n^7 \log^2 n)$

Table 3.2.1. Modulus size and time complexity of Algorithm 3.1 as a function of the parameter m.

Heuristic analysis for HSSP_n^κ

When the components of $\mathbf{h}$ are sums of subsets of fixed Hamming weight we can get a better heuristic bound. Indeed, in such case $\mathbb{E}[\|\mathbf{x}_i\|] \leq \sqrt{m\kappa/n}$. Then, we heuristically have by using (3.9)

$$\det(\mathcal{L}_{\mathbf{x}}^{\perp}) \lesssim \frac{1}{\kappa}\left(\frac{m\kappa}{n}\right)^{\frac{n}{2}}.$$

that combined with the heuristic approximation

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

produces the following analogue of (3.11):

$$\lambda_{m-n}\left(\mathcal{L}_{\mathbf{x}}^{\perp}\right) \lessapprox \sqrt{\gamma_{m-n}}\left(\frac{m\kappa}{n}\right)^{\frac{n}{2(m-n)}} \cdot \kappa^{-\frac{1}{m-n}}. \tag{3.14}$$

and consequently (3.12) becomes

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

This for $m = 2n$ gives us

$$\log Q > 2\iota \cdot n^2 + n\log(n) + \frac{n-2}{2}\log(\kappa) + n.$$

which is smaller for $\kappa \ll n$.

3.2.2 Second Step

The second step of Nguyen-Stern approach consists in applying BKZ to the output basis of the first step and consequently applying a polynomial-time search. In the following, we provide a heuristic analysis of the second step of the Nguyen-Stern algorithm; 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 Section 3.3. Under the proposed model the BKZ block-size β increases almost linearly with n, which implies that the complexity of the attack should be exponential in n.

As done for the heuristic bounds concerning the first step, it is convenient to address separately random instances of HSSP_n and HSSP_n^κ. Indeed, the different algebraic and statistical properties of the two distributions affect the performance of BKZ.

Second Step $\mathbf{HSSP}_n$

In our analysis below, for simplicity we heuristically assume that the lattice $\mathcal{L}_{\mathbf{x}}$ is complete, *i.e.* $\bar{\mathcal{L}}_{\mathbf{x}} = \mathcal{L}_{\mathbf{x}}$. This is consistent with random instances of HSSP_n (Definition 2.2).

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 ι^n should be less than the above gap, which gives the condition:

$$\iota^n \leq \frac{\sqrt{n}}{2} \tag{3.15}$$

We observed in Section 1.1.3 that achieving such a value as Hermite factor ι^n heuristically requires at least $2^{\Omega(\beta)}$ time, by using BKZ reduction with block-size $\beta \simeq n$. Therefore the running time of the Nguyen-Stern algorithm is $2^{\Omega(n)}$, with BKZ block-size $\beta = \omega(n)$ in the second step.

Second Step $\mathbf{HSSP}^{\kappa}_{n}$

In the original BPV generator described in Section 1.6, the size of the subset is fixed; namely, for any i the ith coordinate of $\mathbf{h}$ is the modular scalar product of $\boldsymbol{\alpha}$ and $\tilde{\mathbf{x}} = (x_{1,i}, \ldots, x_{n,i})$, which has Hamming weight κ. Namely, we have a random hidden subset sum problem according to Definition 2.2. Recall $\mathbf{X} \in \{0,1\}^{n \times m}$ is the matrix whose rows are the vectors $\mathbf{x}_i$'s, then $\mathbf{h} = \boldsymbol{\alpha} \cdot \mathbf{X} \pmod{Q}$.

We previously observed that the condition on the Hamming weight implies that

$$\mathbf{1}_n \cdot \mathbf{X} = \kappa \cdot \mathbf{1}_m$$

and then we have that $\mathbf{1}_m \in \bar{\mathcal{L}}_{\mathbf{x}}$. This implies that we have the additional vectors in $\bar{\mathcal{L}}_{\mathbf{x}}$: $\mathbf{y}_i = \mathbf{1}_m - \mathbf{x}_i \in \bar{\mathcal{L}}_{\mathbf{x}}$ for any $i = 1, \ldots, n$.

Moreover, we have $\mathbb{E}[\|\mathbf{x}_i\|^2] = m\kappa/n$ and $\mathbb{E}[\|\mathbf{y}_i\|^2] = m(n-\kappa)/n$, and the vectors $\mathbf{x}_i$'s are not statistically independent. Indeed, given any two coordinates $\tilde{x}_i, \tilde{x}_j$ of $\tilde{\mathbf{x}}$ a generic column of $\mathbf{X}$, we have that $\mathbb{E}[\tilde{x}_i] = \kappa/n$, $\mathbb{E}[\tilde{x}_i^2] = \kappa/n$ and

$$\mathbb{E}[\tilde{x}_i \tilde{x}_j] = \frac{\kappa}{n}\frac{\kappa - 1}{n-1}.$$

This implies that $\mathbb{E}[\langle \mathbf{x}_i, \mathbf{x}_j \rangle]$ is m times such value, hence we obtain

$$\mathbb{E}[\|\mathbf{x}_i - \mathbf{x}_j\|^2] = \mathbb{E}[\|\mathbf{x}_i\|^2] + \mathbb{E}[\|\mathbf{x}_j\|^2] - 2\mathbb{E}[\langle \mathbf{x}_i, \mathbf{x}_j \rangle] = 2m\frac{\kappa}{n}\frac{n-\kappa}{n-1}$$

Similarly, we have that $\mathbb{E}[\tilde{y}_i] = (n-\kappa)/n$, $\mathbb{E}[\tilde{y}_i^2] = (n-\kappa)/n$ and

$$\mathbb{E}[\tilde{y}_i \tilde{y}_j] = \frac{(n-\kappa)}{n}\frac{(n-\kappa)-1}{n-1}.$$

Notice that $\mathbf{y}_i - \mathbf{y}_j = \mathbf{x}_j - \mathbf{x}_i$, then $\mathbb{E}[\|\mathbf{y}_i - \mathbf{y}_j\|^2] = \mathbb{E}[\|\mathbf{x}_i - \mathbf{x}_j\|^2]$. In spite of the fact that $\mathbb{E}[\|\mathbf{x}_i - \mathbf{x}_j\|^2] > \mathbb{E}[\|\mathbf{x}_i\|^2]$ if $\kappa < (n+1)/2$, and $\mathbb{E}[\|\mathbf{x}_i - \mathbf{x}_j\|^2] > \mathbb{E}[\|\mathbf{y}_i\|^2]$ if $\kappa < (n-1)/2$; we still have that the $\mathbf{x}_i - \mathbf{x}_j$ are short vectors. Thus, we have that a "generic" short vector in $\bar{\mathcal{L}}_{\mathbf{x}}$ can be constructed as a binary combination of such vectors. Namely, consider $\mathbf{z} = \sum_{k=1}^{n} b_k \mathbf{z}_k$ with b_j's i.i.d. uniform over $\{0, 1\}$ and $\mathbf{z}_k$ either one of $\mathbf{x}_i - \mathbf{x}_j$. The variance of any component of $\mathbf{z}$ is

$$\mathbb{E}[z_i^2] = \sum_k \mathbb{E}[b_j^2]\mathbb{E}[z_{ki}^2] = \kappa\frac{n-\kappa}{n-1}$$

This implies that the expectation of the square of the norm of $\mathbf{z}$ is $m\kappa\frac{n-\kappa}{n-1}$.

Therefore, we can take as heuristic gap between these generic vectors and the $\mathbf{x}_i$'s as the square root of

$$\frac{m\kappa\frac{n-\kappa}{n-1}}{m\kappa/n} \approx n - \kappa.$$

While the gap between these generic vectors and the $\mathbf{y}_i$'s is heuristically the square root of

$$\frac{m\kappa\frac{n-\kappa}{n-1}}{m(n-\kappa)/n} \approx \kappa.$$

If $\kappa \ll n/2$ the $\mathbf{x}_i$'s are very small, while $\mathbf{y}_i$'s are closer to the generic short vector. Instead, when the problem is balanced, *i.e.* $\kappa \approx n - \kappa \approx n/2$, the $\mathbf{x}_i$'s and $\mathbf{y}_i$'s have the same likelihood of being revealed. The unbalanced case $\kappa \gg n/2$ is symmetric to $\kappa \ll n/2$, interchanging the roles of $\mathbf{x}_i$ and $\mathbf{y}_i$.

Thus, via the same arguments used for HSSP_n we get that to recover the shortest vectors, we should have ι^n smaller than the aforementioned gaps. Therefore, we expect that the running

time of the Nguyen-Stern algorithm is $2^{\Omega(n)}$, with BKZ block-size $\beta = \omega(n)$ in the second step. For the unbalanced case we actually get a slightly lighter condition, as either for the $\mathbf{x}_i$'s or $\mathbf{y}_i$ the gap is smaller; however, the asymptotic complexities remain the same.

In Section 3.1.2 we described a heuristic algorithm to recover the binary vectors given the reduced basis, *i.e.* Algorithm 3.2. This is a greedy strategy that is based on the assumption that the vectors of the reduced basis are a combination of at most two of the $\mathbf{x}_i$'s. In the balanced case we have that we can treat the $\mathbf{y}_i$'s as the $\mathbf{x}_i$'s and expect that the only vectors of the lattice in $\{0, \pm 1\}$ are combination of two such vectors, except the special vector $\mathbf{1}_m$. Then, applying Algorithm 3.2 artificially including $\mathbf{1}_m$ in the list L, we expect to recover all the vectors if $\mathbf{1}_m$ does not belong to the reduced basis. Otherwise, we may have a missing pair. Notice that the vector $\mathbf{1}_m$ has norm $\sqrt{m}$, hence the heuristic gap a generic short vector $\mathbf{z}$ and $\mathbf{1}_m$ is $\sqrt{\kappa \frac{n-\kappa}{n-1}}$. This implies that the likelihood of $\mathbf{1}_m$ appearing in the reduced basis is very similar to the one of either $\mathbf{x}_i$ or $\mathbf{y}_i$, depending on the sparsity. In practice, we observed that $\mathbf{1}_m$ belongs to the bases quite often.

Finally, we remark that in [NS99] the presence of these additional vectors $\mathbf{1}_m$ and $\mathbf{y}_i$'s inside $\bar{\mathcal{L}}_{\mathbf{x}}$ is not mentioned explicitly; neither any method for extracting the fixed Hamming weight basis is illustrated. In the sparse case distinguishing between $\mathbf{x}_i$'s and $\mathbf{y}_i$'s is quite straightforward by using the norm of the vectors, while this is not immediate in the balanced case. Therefore, we propose an algorithm fulfilling this purpose in Section 4.3 an we also describe a variant working if there is a missing pair.

Hermite factors

The target Hermite factor for the second step of the Nguyen-Stern algorithm for HSSP_n is $\gamma = \sqrt{n}/2$, which can be written as $\gamma = a^n$ with $a = (n/4)^{1/(2n)}$. For balanced instances of HSSP_n^κ, we observed that such a value can be taken as a target, too. Table 3.3.1 contains the corresponding target Hermite factors as a function of n.

We ran some experiments on a different lattice, independent from our model of Section 3.1.2, in order to predict the Hermite factor that we can achieve when using BKZ as a function of the block-size β. In particular, we 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 $2^{\iota n}(\det \mathcal{L})^{1/n} = 2^{\iota n} \cdot p^{1/n}$, where $2^{\iota n}$ is the Hermite factor. Table 3.2.2 summarises the results we obtained. The 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.

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

Table 3.2.2. Experimental and simulated Hermite factors for LLL ($\beta = 2$) and for BKZ with block-size β.

Notice that the minimal BKZ block-sizes β required experimentally in Table 3.3.1 to apply Step 2 of Nguyen-Stern, seem coherent with the target Hermite factors from our model, and the experimental Hermite factors from Table 3.2.2. 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 3.2.2, BKZ-10 can achieve 1.015^n, so in Table 3.3.1 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 have 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 3.2.2 it can only achieve 1.013^n. Despite this discrepancy, we believe that our model and the above experiments confirm that the complexity of the Nguyen-Stern algorithm is indeed exponential in n. However, this shows that further investigations regarding the behaviour of BKZ with respect to the lattices arising in this context is crucial for a better understanding.

3.3 Practical experiments

We collect in this section experimental results regarding the Nguyen-Stern attack.

We provide in Table 3.3.1 the result of practical experiments for HSSP_n. The first step consists in the orthogonal lattice attack with two applications of LLL described in Section 3.1.1. For the second step, we receive as input from Step 1 an LLL-reduced basis of the lattice $\mathcal{L}_{\mathbf{x}}$. Table 3.3.1 shows that for $n = 70$ this is sufficient to recover the vectors whose coefficients are in $\{\pm 1, 0\}$ and subsequently to apply then Algorithm 3.2 in order to retrieve the hidden vectors $\mathbf{x}_i$'s. Otherwise, we apply BKZ with block-size $\beta = 10, 20, 30, \ldots$ until we recover the vectors whose coefficients are in $\{\pm 1, 0\}$ and then we apply Algorithm 3.2 to retrieve the $\mathbf{x}_i$'s. 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 $n \geq 170$. We provide the source code of our `SageMath` implementation in `https://github.com/agnesegini/solving_hssp`, based on the L^2 [NS09] and BKZ [CN11] implementations from [fpl16]. For the following experiments we used an Intel Core i7-8665U CPU.

			Step 1		Step 2			
n	m	$\log Q$	LLL $\mathcal{L}_0$	LLL $\mathcal{L}_{\mathbf{x}}^{\perp}$	Hermite	Reduction		Total
70	140	772	2.6 s	1.3 s	1.021^n	LLL	0.8 s	4.7 s
90	180	1151	8.0 s	4.0 s	1.017^n	BKZ-10	1.9 s	14.1 s
110	220	1592	21.9 s	10.0 s	1.015^n	BKZ-20	5.3 s	37.7 s
130	260	2095	52.4 s	19.6 s	1.013^n	BKZ-20	16.8 s	89.6 s
150	300	2659	110.2 s	36.7 s	1.012^n	BKZ-30	127.0 s	5 min
170	340	3282	4 min	76.8 s	1.011^n	BKZ-30	385 min	391 min
190	380	3965	24 min	151.7 s	1.010^n	–	–	–
220	440	5099	81 min	41 min	1.009^n	–	–	–
250	500	6366	173 min	72 min	1.008^n	–	–	–

Table 3.3.1. Running time of the Nguyen-Stern attack for HSSP_n.

Regarding HSSP_n^{κ}, we observed that if $\varsigma = \kappa/n \approx 1/2$, it is difficult to distinguish the vectors $\mathbf{x}_i$'s from the $\mathbf{y}_i$'s. Therefore, in Section 4.3 we will present an algorithm for this purpose, exploiting some statistical properties of the problem. However, this requires slightly larger m, *e.g.* $m = 4n$. In Table 4.3.2 we will present then the result of practical experiments *including* new algorithms present in Chapter 4. While, if the problem is unbalanced, in certain cases the original Nguyen-Stern strategy is sufficient. During our experiment for HSSP_n^{κ} with $\varsigma = \kappa/n \approx 1/4$ we observed that this however works in a few cases; Table 3.3.2 collects their running times.

				Step 1		Step 2		
n	κ	m	$\log Q$	LLL $\mathcal{L}_0$	LLL $\mathcal{L}_{\mathbf{x}}^{\perp}$	Reduction		Total
70	17	140	772	2.4 s	1.1 s	LLL	0.6 s	4.2 s
90	22	180	1151	7.7 s	3.5 s	BKZ-10	2.4 s	13.7 s
110	27	220	1592	20.4 s	8.3 s	BKZ-20	5.6 s	34.6 s
130	32	260	2095	49.7 s	17.0 s	BKZ-20	13.4 s	80.5 s
150	37	300	2659	126.4 s	32.6 s	BKZ-30	82.5 s	4 min
170	42	340	3282	4 min	59.4 s	BKZ-30	158 min	162 min

Table 3.3.2. Running time of the Nguyen-Stern attack applied to instances of $\mathrm{HSSP}_n^{n/4}$.

4

Our algorithm for solving HSSP in polynomial time

Our work aims to enhance the understanding of the hardness of the hidden subset sum problem. We observed in Chapter 2 that for certain choices of parameters, the sample vectors of HSSP are indistinguishable from vectors uniformly sampled from $\mathbb{Z}_Q^m$, *e.g.* if we consider HSSP_n and $\mathrm{HSSP}_n^{n/2}$ with $\log Q \approx n$. Conversely, the work of Nguyen and Stern [NS99] and our analysis expose the weaknesses of other collections of parameters, *e.g.* low-density HSSP. Despite certain issues in the original paper raised during our study of the Nguyen-Stern algorithm for solving the hidden subset sum, this lattice-based attack contains several innovative ideas, which are a valid basis to construct our novel solutions for solving the HSSP. Therefore, in this chapter we propose various algorithms to modify and improve the two-steps attack described in Chapter 3, in order to get a polynomial-time algorithm for solving the hidden subset sum problem.

Road map and reasons

Our rigorous analysis of the orthogonal lattice attack produced precise conditions under which the completion of the lattice $\bar{\mathcal{L}}_{\mathbf{x}}$ generated by the binary vectors $\mathbf{x}_i$'s associate to an instance of HSSP can be computed in polynomial time; see Section 3.2 . Specifically, Theorem 3.1 implies that low-density hidden subset sums are probably weaker, since the lattice $\bar{\mathcal{L}}_{\mathbf{x}}$ can be likely revealed. In Section 4.1 of this chapter we will present our methods for *improving the orthogonal lattice attack*. These improvements have both a theoretical and practical impact, because they allow us to revise the lower bound on the size of the modulus Q, releasing its dependence on the number of samples; moreover, having a better bound leads us to a substantial decrease in time and space complexity when a large number of samples are involved.

Increasing the number of samples has indeed been decisive in solving a further issue displayed by our analysis of the Nguyen-Stern algorithm, *i.e.* retrieving the binary vectors in polynomial time. We remind that in Section 3.2.2 we provided a heuristic analysis of the second step of the Nguyen-Stern algorithm, in which we justify the exponential behaviour of such an algorithm and we argue that using BKZ with a very large block size seems necessary. In

this chapter we propose a novel approach to recover the binary vectors in polynomial time. However, our algorithm requires $m \approx n^2$, instead of $m = 2n$ as the Nguyen-Stern attack. Fundamentally, using vectors of larger dimensions generates a trade-off between the complexity of the orthogonal lattice attack and that one of the second step.

Our new proposal for the second step is not based on a lattice reduction strategy, as it is the initial Nguyen-Stern attack. Namely, our algorithm reduces the problem of disclosing the binary vectors to solving a *multivariate polynomial system*. This algorithm heuristically requires $m \approx n^2$ samples and solves the problem with full asymptotic complexity $\mathcal{O}(n^9)$. Section 4.4 contains experimental results showing that our algorithm performs quite well in practice, as for instance, we can reach $n \simeq 250$ for HSSP_n in a few hours on a single PC.

We remark that, while discerning the kind of random instance we want to solve has a minor impact on the orthogonal lattice attack[1], the different properties of the instances of HSSP_n and HSSP_n^κ affect the second step in a non-trivial way. This is actually reasonable, since the main difference between Definition 2.1 and Definition 2.2 is the generation of the binary vectors $\mathbf{x}_i$'s, which are the target of the second step. We already observed a difference when analysing the behaviour of BKZ in Section 3.2.2, and we will see that the multivariate approach can be adapted in both cases, by using two different strategies.

Finally, as the last contribution of the chapter in Section 4.3 we describe an algorithm to reconstruct the original binary vectors forming the basis with fixed Hamming weight as required for solving HSSP_n^κ.

Later in Chapter 5 and Section 6.2, we will present two further algorithms based on statistical learning for solving the HSSP_n. The first algorithm shows that HSSP_n can be provably solved in polynomial time (using a polynomial number of samples), while the second one is a heuristic variant using fewer samples that is quite efficient in practice. This statistical learning approach based on learning a parallelepiped is only applicable for HSSP_n, essentially because in HSSP_n^κ the $\mathbf{x}_i$'s are not "statistically" independent.

4.1 Improving the orthogonal lattice attack

We have shown that Nguyen-Stern orthogonal lattice attack recovers the hidden lattice $\bar{\mathcal{L}}_{\mathbf{x}}$ in polynomial time with probability at least $1/2$ over the choice of $\boldsymbol{\alpha}$. Both the bound on the size of the modulus Q given by Theorem 3.1 and the complexity of Algorithm 3.1 depend on m, *i.e.* the dimension of the input vector $\mathbf{h}$.

In the following sections we will describe our alternatives to Nguyen-Stern's second step. These algorithms will require more samples than in [NS99], namely at least $m \gtrsim n^2/2$ samples instead of $m = 2n$. This implies 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 at least $m \simeq n^2/2$ instead of

[1] See Section 3.2.1.

$m = 2n$. Then, naively applying Algorithm 3.1 would lead to higher bounds for the moduli and complexities.

Therefore, we now first explain how to modify Nguyen-Stern's first step 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 m, we apply LLL in parallel on m/t square matrices of dimension $t = 2n$, which is much faster. Eventually, we show in Section 4.1.2 that a single application of LLL is sufficient.

We remark that improving the first step has an impact that is both practical and theoretical. Indeed, it implies that the minimal size of the modulus for which we can disclose the hidden lattice (thus, solving the hidden subset sum problem) does not depend on the value of m.

4.1.1 Blocks orthogonal lattice attack

In this section, we show how to adapt the first step, *i.e.* the orthogonal lattice attack from [NS99] recalled in Section 3.1.1, 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 $m = (b+1) \cdot n$ for $b > 0$, while applying LLL on matrices of dimension $t = 2n$ only. Notice that b does not have to be constant, *e.g.* one can consider $b = n$.

As illustrated in Figure 4.1, this is relatively straightforward: we apply Algorithm 3.1 from Section 3.1.1 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 $\mathbf{h} = [\mathbf{h}_0, \ldots, \mathbf{h}_b]$

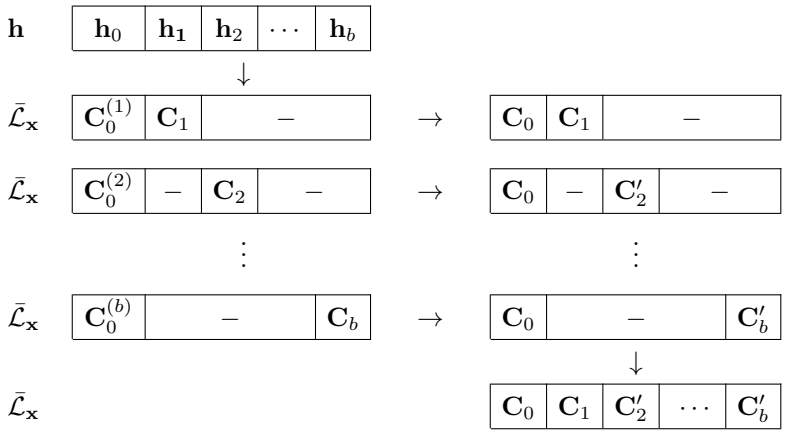


Fig. 4.1. Computation of a lattice basis of $\bar{\mathcal{L}}_{\mathbf{x}}$ by blocks.

where $m = (b+1) \cdot n$ and $\mathbf{h}_i \in \mathbb{Z}^n$, we apply Algorithm 3.1 on each of the b sub-vectors of the form $(\mathbf{h}_0, \mathbf{h}_i) \in \mathbb{Z}^{2n}$ for $1 \leq i \leq b$. For each $1 \leq i \leq b$ this gives us $\mathbf{C}_0^{(i)} \| \mathbf{C}_i \in \mathbb{Z}^{n \times 2n}$, the completion 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 Figure 4.1 for an illustration. This gives Algorithm 4.1 below. We denote Algorithm 3.1 from Section 3.1.1 by OrthoLat.

Algorithm 4.1 Orthogonal lattice attack with $m = (b+1) \cdot n$ samples

Input: $\mathbf{h} \in \mathbb{Z}^m, Q, n, m = (b+1) \cdot n$.
Output: A basis of $\bar{\mathcal{L}}_{\mathbf{x}}$.

1: Write $\mathbf{h} = [\mathbf{h}_0, \ldots, \mathbf{h}_b]$ where $\mathbf{h}_i \in \mathbb{Z}^n$ for all $0 \le i \le b$.
2: **for** $i \leftarrow 1$ to b **do**
3: $\quad \mathbf{y}_i \leftarrow [\mathbf{h}_0, \mathbf{h}_i]$
4: $\quad \mathbf{C}_0^{(i)} \| \mathbf{C}_i \leftarrow \mathsf{OrthoLat}(\mathbf{y}_i, Q, n, 2n)$
5: $\quad \mathbf{Q}_i \leftarrow \mathbf{C}_0^{(1)} \cdot (\mathbf{C}_0^{(i)})^{-1}$
6: $\quad \mathbf{C}'_i \leftarrow \mathbf{Q}_i \cdot \mathbf{C}_i$
7: **end for**
8: $\mathbf{C}_0 = \mathbf{C}_0^{(1)}$
9: **return** $[\mathbf{C}_0, \mathbf{C}_1, \mathbf{C}'_2, \cdots, \mathbf{C}'_b]$

A minor difficulty is that in principle, when applying OrthoLat (Algorithm 3.1) 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 3.1 with a certain projection π we recover the completion $\overline{\pi(\mathcal{L}_{\mathbf{x}})}$. Hence, if 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 4.1, let us first 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 $\mathbf{Q}_i = \mathbf{C}_0^{(1)} \cdot (\mathbf{C}_0^{(i)})^{-1}$ for $2 \le i \le b$ must be integral.

Proposition 4.1. *Let $m = (b+1) \cdot n$ for $b \in \mathbb{N}$ and $b > 0$. Assume that the projection of the lattice $\mathcal{L}_{\mathbf{x}} \subseteq \mathbb{Z}^m$ 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 4.1 recovers a basis of $\bar{\mathcal{L}}_{\mathbf{x}}$ in polynomial time, assuming that Q is a prime of bitsize at least $4n^2(\log n + 1)$.*

Proof. As first task, we show that if Q is a prime of bitsize at least $4n^2(\log n + 1)$, with probability at least $1/2$ over $\boldsymbol{\alpha}$, the orthogonal lattice attack recovers a basis $\mathbf{C}_0^{(i)} \| \mathbf{C}_i$ of the completed projection of $\mathcal{L}_{\mathbf{x}}$ to the first n coordinates and the $(i+1)$th subset of n coordinates for each $1 \le i \le b$. Following the proof of Theorem 3.1, this happens if

$$\sqrt{2n^2} \cdot 2^n \cdot (2n)^{2n} < \lambda_1 \left(\Lambda_Q^{\perp}(\boldsymbol{\alpha}) \right).$$

Since this condition does not depend on the projection, a single application of Lemma 1.16 is sufficient to prove our claim.

Let us denote by $\mathbf{X}$ the basis 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 $i = 1, \ldots, b$ 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 $i = 1, \ldots, b$. 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 $\mathbf{C}_0 \coloneqq \mathbf{C}_0^{(1)}$ as projection on the first n coordinates. This implies that $[\mathbf{C}_0, \mathbf{C}_1, \mathbf{C}'_2, \cdots \mathbf{C}'_b]$ is a basis of $\bar{\mathcal{L}}_{\mathbf{x}}$.

□

For random instances of HSSP_n we observed that the condition on the completeness was always verified in our experiments. Conversely, we know that it can never hold true for random instances of HSSP_n^κ. Indeed, we observed in Section 3.2.1 that $\kappa \mathbf{1}_m \in \mathcal{L}_{\mathbf{x}}$ and that $\mathcal{L}_{\mathbf{x}} \subseteq \mathcal{L}_1 \subseteq \bar{\mathcal{L}}_{\mathbf{x}}$, where $\mathcal{L}_1$ is the lattice generated by $\mathbf{1}_m, \mathbf{x}_1, \ldots, \mathbf{x}_n$. Assuming $\mathbf{1} \notin \mathcal{L}_{\mathbf{x}}$, this implies that $\pi(\mathcal{L}_{\mathbf{x}}) \subsetneq \pi(\mathcal{L}_1) \subseteq \pi(\bar{\mathcal{L}}_{\mathbf{x}})$. For each set of coordinates we have $\pi(\mathbf{1}_m) = \mathbf{1}_{2n}$. Then, substituting in the arguments above $\mathcal{L}_1$ to $\mathcal{L}_{\mathbf{x}}$, we obtain a similar statement. This proves the following Theorem 4.2.

Theorem 4.2. *Let $m = (b+1) \cdot n$ for $b \in \mathbb{N}$ and $b > 0$. Assume that the projection of the lattice $\mathcal{L}_{\mathbf{x}} \subseteq \mathbb{Z}^m$ over the first n components has rank n, and that either the projection of $\mathcal{L}_{\mathbf{x}}$ or $\mathcal{L}_1$ over the first $2n$ coordinates is complete. With probability at least $1/2$ over the choice of $\boldsymbol{\alpha}$, Algorithm 4.1 recovers a basis of $\bar{\mathcal{L}}_{\mathbf{x}}$ in polynomial time, assuming that Q is a prime of bitsize at least $4n^2(\log n + 1)$.*

For random instances of HSSP_n^κ the latter assumption was always true for the instances analysed during our experiments.

Heuristic analysis

Regarding the size of the modulus Q, 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 Section 3.2.1, namely

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

with $\iota = 0.035$. The time complexity of Algorithm 4.1 is dominated by the cost of applying OrthoLat (Algorithm 3.1) to each $\mathbf{y}_i$, which is heuristically $\mathcal{O}(n^9)$ from Section 3.2.1. Notice, the choice $t = 2n$ is then optimal according to Table 3.2.1.

Therefore, the heuristic complexity of Algorithm 4.1 is $b \cdot \mathcal{O}(n^9) = \mathcal{O}(m \cdot n^8)$. In particular, for $m \simeq n^2/2$, the heuristic complexity of Algorithm 4.1 is $\mathcal{O}(n^{10})$, instead of $\mathcal{O}(n^{16})$ with the naive method (see Table 3.2.1). Finally, notice that Algorithm 4.1 can be performed in parallel starting from the second iteration.

In Section 4.1.2 we will describe an improved algorithm with complexity $\mathcal{O}(n^9)$.

Remark 4.3. Theorem 4.2 implies that under given hypotheses the lower bound on the modulus Q does not depend on the number of samples, but rather on the density. Namely, $d = \mathcal{O}(1/(n \log n))$ and heuristically $d = \mathcal{O}(1/n)$ is sufficient to solve the problem for any m. Notice that these hypotheses are sufficient but not necessary since, for example, one could state a similar result using different coordinates for the projection. Alternatively, a sufficient condition is also only asking that for at least a projection either $\overline{\pi(\mathcal{L}_{\mathbf{x}})} = \pi(\bar{\mathcal{L}}_{\mathbf{x}})$ or $\overline{\pi(\mathcal{L}_1)} = \pi(\bar{\mathcal{L}}_1)$.

4.1.2 Improvement via size reduction

In Section 4.1.1, we showed a method to compute $\bar{\mathcal{L}}_{\mathbf{x}}$ by $\mathcal{O}(m/2n)$ parallel applications of LLL. The overall heuristic time complexity is $\mathcal{O}(n^{10})$ for $m \approx n^2$.

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 where $m = 2n$.

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 4.4 (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 4.2 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 4.5 (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 $\mathbf{u} \in \mathcal{L}$.*

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 Q 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 Q, 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 the 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 Figure 4.2 for an illustration. This size-reduction is much faster than repeatedly applying LLL as in Section 4.1.1. We describe the corresponding algorithm below.

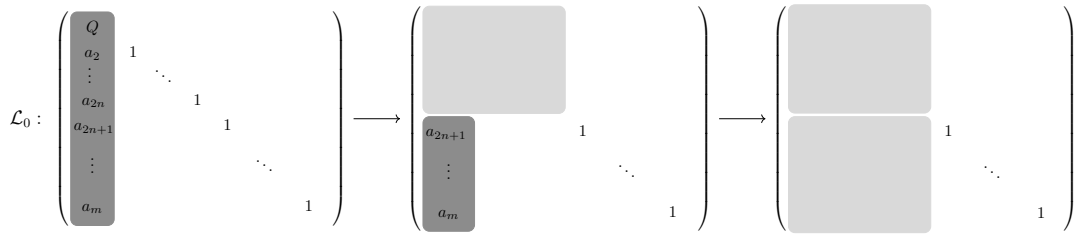


Fig. 4.2. In the initial basis matrix the components of the first column are large. Then by applying LLL on the $2n \times 2n$ submatrix the corresponding components become small; this already gives n orthogonal vectors in $\mathcal{L}_{\mathbf{x}}^{\perp}$. Then by size-reducing the remaining $m - 2n$ rows, one obtains small components on the $2n$ columns, and therefore $m - 2n$ additional orthogonal vectors. In total we obtain $m - n$ orthogonal vectors.

The following lemma shows that under certain conditions on the lattice $\mathcal{L}_{\mathbf{x}}^{\perp}$, Algorithm 4.3 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.

Lemma 4.6. *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 $1 \le i \le n$ with $\mathbf{c}_i \in \mathbb{Z}^{2n}$ and $\|\mathbf{c}_i\| \le B$, 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 $2n + 1 \le i \le m$ with $\mathbf{c}_i \in \mathbb{Z}^{2n}$ and $\|\mathbf{c}_i\| \le B$. Then if $(\gamma B + 1)\sqrt{mn} \le \lambda_1\left(\Lambda_Q^{\perp}(\boldsymbol{\alpha})\right)$ where $\gamma = 1 + 4n(9/2)^n$, Algorithm 4.3 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 $1 \le i \le n$, and $m - 2n$ vectors $\mathbf{a}'_i \in \mathcal{L}_{\mathbf{x}}^{\perp}$ for $2n + 1 \le i \le m$.*

Proof. We let $\mathcal{L} \subset \mathbb{Z}^{2n}$ be the lattice generated by the LLL-reduced basis of vectors $\mathbf{a}_1, \ldots, \mathbf{a}_{2n}$. By assumption the lattice $\mathcal{L}$ contains n linearly independent vectors $\mathbf{c}_i$ with $\|\mathbf{c}_i\| \le B$. Therefore we must have for all $1 \le i \le n$:

Algorithm 4.3 Fast generation of orthogonal vectors

Input: $\mathbf{h} \in \mathbb{Z}^m$, Q, 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 Q, 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 $\mathbf{t}_i = [-h_i h_1^{-1}[Q]\ 0\ \cdots\ 0] \in \mathbb{Z}^{2n}$.
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 $2n+1 \leq i \leq m$.

$$\|\mathbf{a}_i\| \leq 2^n \cdot \lambda_n(\mathcal{L}) \leq 2^n B \leq \gamma B$$

This implies $\|\mathbf{a}'_i\| = \|\mathbf{a}_i\| \leq \gamma B$ which gives $\|\mathbf{a}'_i\|\sqrt{mn} \leq \lambda_1\left(\Lambda_Q^{\perp}(\boldsymbol{\alpha})\right)$ for all $1 \leq i \leq n$; this corresponds to Condition 3.3 in Lemma 3.2. Therefore we must have $\mathbf{a}'_i \in \mathcal{L}_{\mathbf{x}}^{\perp}$ for all $1 \leq i \leq n$.

For $2n+1 \leq i \leq m$, we consider the vector $\mathbf{t}_i = [-h_i h_1^{-1}[Q]\ 0\ \cdots\ 0] \in \mathbb{Z}^{2n}$, and the vector $\mathbf{t}'_i = [\mathbf{t}_i\ 0 \cdots 1 \cdots 0] \in \mathbb{Z}^m$ where the 1 component is at index i. We have $\mathbf{t}'_i \in \mathcal{L}_0$. Since by assumption there exists a vector $\mathbf{c}'_i = [\,\mathbf{c}_i\ 0\ \cdots\ 1\ \cdots\ 0\,] \in \mathcal{L}_{\mathbf{x}}^{\perp} \subset \mathcal{L}_0$, we must have $\mathbf{t}'_i - \mathbf{c}'_i = [\mathbf{t}_i - \mathbf{c}_i\ 0 \cdots 0] \in \mathcal{L}_0$, and therefore $\mathbf{u} := \mathbf{t}_i - \mathbf{c}_i \in \mathcal{L}$.

Let $\mathbf{v} \in \mathcal{L}$ be the vector obtained from Babai's rounding at Step 6, when given the vector $\mathbf{t}_i$ as input. Since the lattice basis $\mathbf{a}_1, \ldots, \mathbf{a}_{2n} \in \mathbb{Z}^{2n}$ is LLL-reduced, from Theorem 4.5 we must have:

$$\|\mathbf{t}_i - \mathbf{v}\| \leq \gamma \|\mathbf{t}_i - \mathbf{u}\| = \gamma \|\mathbf{c}_i\| \leq \gamma B$$

where $\gamma = 1 + 4n(9/2)^n$. Therefore letting $\mathbf{a}'_i = [(\mathbf{t}_i - \mathbf{v})\ 0\ \ 1\ \ 0] \in \mathbb{Z}^m$, we must have $\mathbf{a}'_i \in \mathcal{L}_0$ and moreover $\|\mathbf{a}'_i\| \leq \|\mathbf{t}_i - \mathbf{v}\| + 1 \leq \gamma B + 1$. As previously this implies $\|\mathbf{a}'_i\|\sqrt{mn} \leq \lambda_1\left(\Lambda_Q^{\perp}(\boldsymbol{\alpha})\right)$ and therefore we must have $\mathbf{a}'_i \in \mathcal{L}_{\mathbf{x}}^{\perp}$ for all $2n+1 \leq i \leq m$. □

Complexity analysis

We assume for simplicity $m \approx n^2$, as this will be the value of m used by our algorithms later described in this work. Moreover, since the approximation factor γ for CVP is similar to the LLL Hermite factor, we use the same modulus size as previously, *i.e.* $\log Q \simeq 2\iota n^2 + n \cdot \log n$ with $\iota = 0.035$. Then, as in Section 3.2.1 the complexity of the first LLL reduction with L^2 is $\mathcal{O}(n^5 \log^2 Q) = \mathcal{O}(n^9)$.

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 $\log Q$ bits of precision; this has to be done only once, and takes $\mathcal{O}(n^3 \log^2 Q) = \mathcal{O}(n^7)$ time. Then for each Babai's rounding we need one vector

matrix multiplication, with precision $\log Q$ bits. Since the vector has actually a single non-zero component, the complexity is $\mathcal{O}(n \log^2 Q) = \mathcal{O}(n^5)$. With $m = \mathcal{O}(n^2)$, the total complexity of size-reduction is therefore $\mathcal{O}(n^7)$. In Section 4.1.2, we will 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 4.3 for computing a generating set of $\mathcal{L}_{\mathbf{x}}^{\perp}$ is therefore $\mathcal{O}(n^9)$, instead of $\mathcal{O}(n^{10})$ in Section 4.1.1.

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 Figure 4.3, 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.

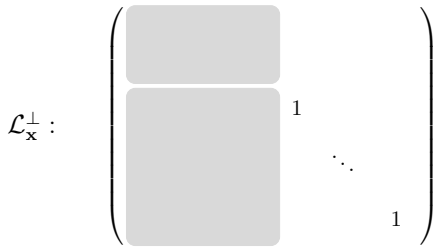


Fig. 4.3. Structure of the generating set of $\mathcal{L}_{\mathbf{x}}^{\perp}$.

More precisely, from Algorithm 4.3, 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:

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

where $\mathbf{U} \in \mathbb{Z}^{n\times 2n}$ and $\mathbf{V} \in \mathbb{Z}^{(m-2n)\times 2n}$. As in Section 3.1.1, using the LLL-based algorithm from [NS97] recalled in Sections 1.2, 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 $\bar{\mathcal{L}}_{\mathbf{x}} = (\mathcal{L}_{\mathbf{x}}^{\perp})^{\perp}$.

Optimization of the size-reductions

We describe here a heuristic faster algorithm for the size-reductions, with complexity $\mathcal{O}(n^{20/3})$ instead of $\mathcal{O}(n^7)$. The idea is to first apply LLL to the $(n/k) \times (n/k)$ upper-left submatrix of the lattice $\mathcal{L}_0$. Then the vectors are first size-reduced with respect to this small submatrix. Then LLL is applied to the $(2n) \times (2n)$ upper-left submatrix as previously, and the vectors are then size-reduced with respect to this larger submatrix.

The advantage is that for the first size-reduction, we are now working with a matrix of dimension n/k instead of $2n$. And in the second size-reduction, when working with the full $2n \times 2n$ matrix, we start with vectors with much smaller components, of size roughly $(k/n) \log Q$ instead of $\log Q$. Below we show that taking $k = n^{1/3}$ is optimal.

For the first size-reduction, as previously, one must first invert once a $(n/k) \times (n/k)$ matrix with $\log Q$ bits of precision, which takes $\mathcal{O}((n/k)^3 \log^2 Q)$ time. The complexity of size-reducing each vector is then $\mathcal{O}((n/k) \log^2 Q)$. Then the vectors have components of size roughly $(k/n) \log Q$ bits. Therefore for the second size-reduction one must first invert a $2n \times 2n$ matrix with $(k/n) \log Q$ bits of precision, which takes $\mathcal{O}(n^3((k/n) \log Q)^2) = \mathcal{O}(nk^2 \log^2 Q)$. For each vector the second size reduction has complexity $\mathcal{O}(n^2((k/n) \log Q)^2) = \mathcal{O}(k^2 \log^2 Q)$.

We see that for $m = \mathcal{O}(n^2)$ vectors to size-reduce, the cost of the two matrix inversions is dominated by the cost of the size reductions, which take $\mathcal{O}(n^2(n/k) \log^2 Q)$ and $\mathcal{O}(n^2 k^2 \log^2 Q)$ respectively. Therefore, the optimal value is $n/k = k^2$, which gives $k = n^{1/3}$. The time complexity of the size-reduction step for the$m = \mathcal{O}(n^2)$ vectors is then $\mathcal{O}(n^{20/3})$ instead of $\mathcal{O}(n^7)$. Notice that in practice it is more efficient to size-reduce multiple vectors at a time, in order to have a single matrix-matrix multiplication, rather than a sequence of vector-matrix multiplications.

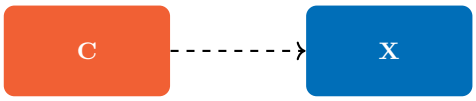
4.2 Our multivariate approach

Recall that the Nguyen-Stern attack for solving HSSP is divided in the two following steps.

1. From the sample vector $\mathbf{h}$ and the modulus Q, 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 Q, recover the weights α_i.

In the previous chapter we have argued that the complexity of the second step of the Nguyen-Stern attack is exponential in n, since BKZ with a large block size should be used to retrieve the binary vectors.

In this section we describe an alternative second step with heuristic 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 3.1 from Section 3.1.1 to the vector $\mathbf{h}$ of dimension $m \simeq n^2/2$. But for $n \simeq 200$ we 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 Q would need to be much larger due to the Hermite factor of LLL in such a large dimension (see Table 3.2.1). However, applying our improvements to the orthogonal lattice attack described in Section 4.1 we can compute a lattice basis of $\bar{\mathcal{L}}_{\mathbf{x}} \subset \mathbb{Z}^m$ for large m with the same asymptotic complexity and by using the same modulus size $\log Q$ as for $m = 2n$. Although our improved algorithm requires extra assumptions on the lattice $\mathcal{L}_{\mathbf{x}}$, we found that such hypotheses are always verified in practice by any random instance of the problem.

The output of the first step is a lattice containing the target vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n$; thus, the goal of the second step is to recover these vectors given as input such a lattice. The technique we propose in this section reduces the binary vectors' search to solving a *multivariate quadratic polynomial system*. Multivariate systems are a widely studied topic with several cryptographic applications. Indeed, they are both directly and indirectly linked to the security of a large number of constructions. Namely, since the task of solving non-linear multivariate polynomial system is difficult in general, this problem has been taken as underlying assumption for encryptions and signatures, *e.g.* [Pat96, DS05]. Moreover, multivariate polynomial systems have proven themselves to be a very powerful tool for cryptanalysis. Indeed, they have been used to build attacks both for explicitly related protocols [CKPS00, Pat95, BFP12, Cou01, FJ03, KS98, KS99] and for constructions based on other assumptions, such as *code-based* [FOP+16, FGUO+11] and *LWE-based* [AG11, ACFP14].

In the first part of this section, we explain how to produce a convenient quadratic multivariate polynomial system given the output of the orthogonal lattice attack, both for random instances of HSSP_n and HSSP_n^κ. Then, the latter part is dedicated to a discussion regarding the solvability of such a system. Specifically, we describe our *ad hoc* algorithms based on a joint strategy of linearization and either eigenspace or half-space exploitation.

The bounds on m and the complexity we obtain by defining our own method for solving HSSP-systems are comparable to those we could get by using known methods, *e.g.* Gröbner bases [CLO98]. However, we found that our algorithm is actually more practical and easy to integrate with the orthogonal lattice attack, since both it exploits the structure and the peculiarities of the system, and can be described in terms of linear algebra only. Moreover, being self-contained, it makes the final result on the hardness of the hidden subset sum problem more comprehensible and accessible for those who are not familiar with computational algebraic geometry. Experimental results are collected in Section 4.4.

4.2.1 Deriving a quadratic multivariate polynomial system

Our new algorithm belongs to the family of *two-steps* algorithms introduced in Chapter 2 and it respects the overall structure pictured by Fig. 2.1. The first step, *i.e.* the (improved) orthogonal lattice attack, recovers 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 retrieve the original vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \bar{\mathcal{L}}_{\mathbf{x}}$, namely to solve the following problem:

Problem 4.1. *Let* $\mathbf{X} \in \{0,1\}^{n \times m}$. *Given* $\mathbf{C} \in \mathbb{Z}^{n \times m}$ *such that* $\mathbf{WC} = \mathbf{X}$ *for some* $\mathbf{W} \in \mathbb{Z}^{n \times n} \cap \mathrm{GL}_n(\mathbb{Q})$, *recover* $\mathbf{W}$ *and* $\mathbf{X}$.

For any target row vector $\mathbf{x}_i \in \{0,1\}^m$ there exists a row vector $\mathbf{w}_i \in \mathbb{Z}^n$ such that $\mathbf{w}_i \cdot \mathbf{C} = \mathbf{x}_i$. The crucial observation is that since all components of the vectors $\mathbf{x}_i$ are binary, they must all satisfy the quadratic equation

$$y^2 - y = 0.$$

Now, denoting by $\tilde{\mathbf{c}}_1, \ldots, \tilde{\mathbf{c}}_m$ the column vectors of $\mathbf{C}$, we have

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

Therefore, for each $i = 1, \ldots, n$ we have:

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

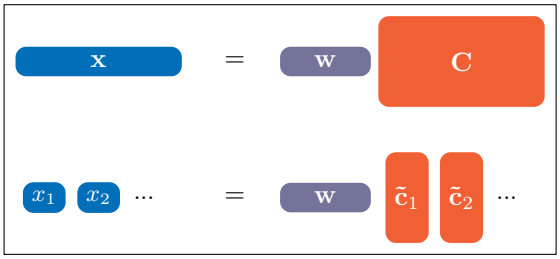


Fig. 4.4. Disassembling the row vectors $\mathbf{x}_i$'s.

Let $\mathbf{w} = (w_1, \ldots, w_n)$ be a row vector of variables, for each column of $\mathbf{C}$ we can define a quadratic polynomial $\langle \mathbf{w}, \tilde{\mathbf{c}}_j \rangle^2 - \langle \mathbf{w}, \tilde{\mathbf{c}}_j \rangle \in \mathbb{Z}[\mathbf{w}]$. Then, the vectors $\mathbf{w}_1, \ldots, \mathbf{w}_n$ are among the solutions over $\mathbb{Z}$ of the induced quadratic polynomial multivariate system of m equations:

$$\begin{cases} \mathbf{w} \cdot \tilde{\mathbf{c}}_1 \tilde{\mathbf{c}}_1^\intercal \cdot \mathbf{w}^\intercal - \mathbf{w} \cdot \tilde{\mathbf{c}}_1 = 0 \\ \qquad \vdots \\ \mathbf{w} \cdot \tilde{\mathbf{c}}_m \tilde{\mathbf{c}}_m^\intercal \cdot \mathbf{w}^\intercal - \mathbf{w} \cdot \tilde{\mathbf{c}}_m = 0 \end{cases} \tag{4.1}$$

Notice that in order to recover the binary vectors $\mathbf{x}_i$'s is sufficient solve the system, since given $\mathbf{w}_i$ we can compute $\mathbf{w}_i \cdot \mathbf{C} = \mathbf{x}_i$. Therefore, our multivariate approach essentially consists in constructing and solving such a system.

Remark 4.7. In our framework, we presume that the system derived from HSSP is overdetermined, since $m > n$. According to the literature, we can expect to solve a "random" overdetermined quadratic systems in polynomial time if $m \gtrsim n^2$ by using Gröbner basis techniques for zero dimensional ideals [CLO98, BFSY, BFP12, Fau99, Fau02]. This bound applies also for the XL algorithm [CKPS00], which however is designed supposing that the system has a unique solution. This is not the case for our system (4.1). Indeed, depending on the distribution of HSSP and the parameters, we will see that a precise number of solutions is predictable. More generally, considering these systems as "random" appears not entirely correct, as they inherit some structure from the main problem.
Moreover, the aforementioned methods for solving multivariate polynomial systems revealed themselves inefficient in practice for this specific application. Therefore, in the following we will discuss how to solve (4.1), taking advantage of our knowledge of the general setting. Eventually, we will show that for $m \simeq n^2/2$ the hidden subset sum problem can be heuristically solved in polynomial time, using a multivariate approach. We could not substantially decrease this value by using this explicit approach, but this is, as a matter of fact, coherent with the cited literature.

4.2.2 Recovering the binary vectors $\mathbf{x}_i$'s

Our algorithm for solving system (4.1) performs three main operations: a linearization, a kernel computation and finally a driven inspection of such a kernel. While the first and the second phases are pretty straightforward, the latter is highly affected by the distribution that is used to generate the binary vectors of the HSSP instance we are solving.

Linearization and kernel computation

Linearization techniques for solving systems have been used in literature since [Laz83]. For instance, the XL algorithm [CKPS00] is a method widely used for cryptanalysis and based on this principle. We briefly recall how to apply linearization to system (4.1).

Given $\mathbf{C}$ the basis of $\bar{\mathcal{L}}_{\mathbf{x}}$. Since the coordinates of its columns are $(\tilde{\mathbf{c}}_j)_i = \mathbf{C}_{ij}$, for all $1 \leq j \leq m$, we can write:

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

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

$$\mathbf{r}_j = ((2 - \delta_{i,k})\mathbf{C}_{ij}\mathbf{C}_{kj})_{1 \leq i \leq k \leq n} \in \mathbb{Z}^{\frac{n^2+n}{2}}. \tag{4.2}$$

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

$$\mathbf{E} = \left[\frac{\mathbf{R}}{-\mathbf{C}} \right] \in \mathbb{Z}^{\frac{n^2+3n}{2} \times m}.$$

Hence, (4.1) is equivalent to

$$\begin{cases} [\ \mathbf{z} \mid \mathbf{w}\ ] \cdot \mathbf{E} = 0 \\ \mathbf{z} = (w_i w_k)_{1 \leq i \leq k \leq n} \in \mathbb{Z}^{\frac{n^2+n}{2}} \end{cases} \tag{4.3}$$

This implies that the solution of the system is the intersection of $\ker \mathbf{E}$, *i.e.* the left kernel of $\mathbf{E}$, and the set

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

Therefore, our method consists in first computing $\ker \mathbf{E}$, and subsequently we examine it for computing vectors in such intersection, relying on the heuristic assumption that the matrix $\mathbf{E}$ has the maximal "possible" rank[2] when $m > (n^2 + n)/2$. We observed that this is actually always verified in practice.

Solving HSSP$_n$ via eigenspace computation

For a random instance of HSSP$_n$, we expect that the unique non trivial solutions are $\mathbf{w}_1, \ldots, \mathbf{w}_n$, 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 either 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.

Consider the set of vectors

[2] This depends on the number of solutions. See Section 4.2.3.

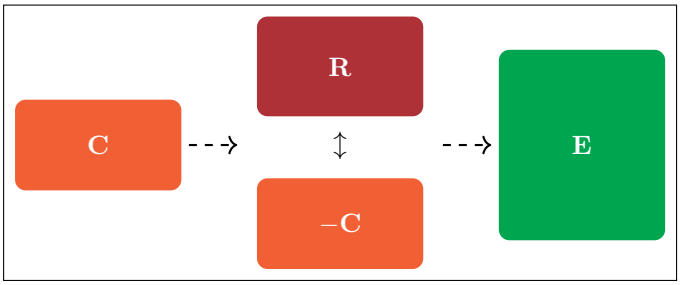


Fig. 4.5. Construction of $\mathbf{E}$.

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

Since by assumption the vectors $\mathbf{w}_i$'s are linearly independent, $\mathrm{Span}(\mathcal{W})$ is a subspace of dimension n of $\ker \mathbf{E}$. This implies that $\dim \ker \mathbf{E} \ge n$.

Heuristic 1. *If $m > (n^2+n)/2$, the matrix $\mathbf{R}$ has rank $(n^2+n)/2$.*

Heuristic 1 implies then $\operatorname{rank} \mathbf{E} \ge (n^2+n)/2$. So $\dim \ker \mathbf{E} = n$. Since the set of n vectors in $\mathcal{W}$ forms then 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:

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

First, note that the last n components in the vectors in $\mathcal{W}$ correspond to the linear part in the quadratic equations of (4.1). So, we can 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:

$$\mathbf{K} = \big[\, \mathbf{M} \mid \mathbf{I}_n \big] \tag{4.4}$$

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 (4.3) 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 $1 \leq i \leq n$.

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

Eigenspaces computation

Consider the first coordinates $w_{j,1}$ of the vectors $\mathbf{w}_j$, for example. 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 $\mathbf{M}_1$; specifically, 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 $1 \leq k \leq s$, 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 4.4.

To better understand 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; so, that branch of the algorithm can be closed. See Fig. 4.6 for an illustration.

Algorithm 4.4 Multivariate attack for HSSP_n

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 $i = 1, \ldots, n$.

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 $1 \le j \le m$.
$\mathbf{E} = \left[\begin{array}{c} \mathbf{r}_1 \cdots \mathbf{r}_m \\ \hline -\mathbf{C} \end{array}\right] \in \mathbb{Z}^{\frac{n^2+3n}{2} \times m}$
$\mathbf{K} \leftarrow$Ker $\mathbf{E}$ with $\mathbf{K} = \left[\begin{array}{c|c} \mathbf{M} & \mathbf{I}_n \end{array}\right] \in \mathbb{Q}^{n \times \frac{n^2+3n}{2}}$
Write $\mathbf{M} = [\tilde{\mathbf{m}}_{ik}]_{1 \le i \le k \le n}$ where $\tilde{\mathbf{m}}_{ik} \in \mathbb{Q}^n$.
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 $i > k$.
$L \leftarrow [\mathbf{I}_n]$
for $i \leftarrow 1$ to n **do**
    $L_2 \leftarrow []$
    for all $\mathbf{V} \in L$ **do**
        if rank $\mathbf{V} = 1$ **then**
            Append a generator $\mathbf{v}$ of $\mathbf{V}$ to L_2.
        else
            Compute $\mathbf{A}$ such that $\mathbf{V} \cdot \mathbf{M}_i = \mathbf{A} \cdot \mathbf{V}$.
            Append all eigenspaces $\mathbf{U}$ of $\mathbf{A}$ to L_2.
        end if
    end for
    $L \leftarrow L_2$
end for
$X \leftarrow []$
for all $\mathbf{v} \in L$ **do**
    Find $c \neq 0$ such that $\mathbf{x} = c \cdot \mathbf{v} \cdot \mathbf{C} \in \{0,1\}^m$, and append $\mathbf{x}$ to X.
end for
return X

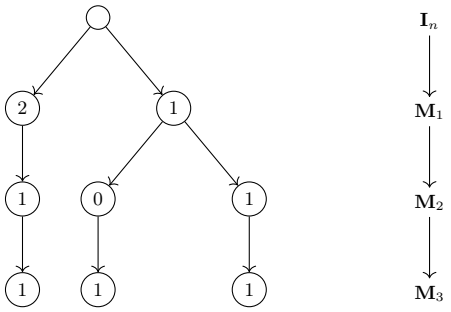


Fig. 4.6. An example of the tree we obtain for $\mathbf{w_1} = (2,1,1), \mathbf{w_2} = (1,0,1), \mathbf{w_3} = (1,1,1)$. The matrix $\mathbf{M}_1$ has an eigenspace of dimension 1 $E_{1,2}$ and one of dimension 2 $E_{1,1}$. At the first iteration we obtain therefore $\mathbf{w_1}$. Then we compute the restriction of $\mathbf{M}_2$ to $E_{1,1}$; this has two distinct eigenvalues 0 and 1, which enables us to recover the eigenvectors $\mathbf{w_2}$ and $\mathbf{w_3}$. All the nodes at depth 2 represent 1-dimensional spaces, hence the algorithm terminates.

Analysis and improvement via reduction modulo p

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 $m = \mathcal{O}(n^2)$, this requires $\mathcal{O}(n^6)$ arithmetic operations. Thus we have shown:

Lemma 4.8. *Let* $\mathbf{C}$ *be a base matrix of the lattice* $\bar{\mathcal{L}}_{\mathbf{x}}$ *associated to a random instance of* $\mathrm{HSSP}_n(m, Q)$. *Let* $\mathbf{R} \in \mathbb{Z}^{\frac{n^2+n}{2} \times m}$ *the matrix whose columns are the* $\mathbf{r}_i$ *constructed as in (4.2). If* $\mathbf{R}$ *has rank* $\frac{n^2+n}{2}$, *then the vectors* $\mathbf{x}_i$*'s can be recovered by performing* $\mathcal{O}(n^6)$ *arithmetic operations.*

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 coefficients of the input matrix $\mathbf{C}$ can be very large, causing the arithmetic operations to be extremely slow. Therefore, in practice it is more efficient to work modulo a prime p instead of over $\mathbb{Q}$.

Indeed, Problem 4.1 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 (4.1) 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, $p = 3$ and $m = (n^2 + 4n)/2$ were sufficient to recover the original vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n$. The heuristic time complexity is $\mathcal{O}(n^6)$, while the space complexity is $\mathcal{O}(n^4)$. See Section 4.4.1 for our practical experiments.

Solving $\mathrm{HSSP}^{\kappa}_{n}$ via bit guessing

Analysing the hardness of random $\mathrm{HSSP}^{\kappa}_{n}$, we have to take into account the relation between the parameters n and κ. If the problem is highly unbalanced, a brute force approach can be more convenient in terms of overall time and space complexity, *e.g.* if $\binom{n}{\kappa} = \mathsf{poly}(n)$. Conversely, this is not very efficient if κ is too close to $n/2$. However, similarly to HSSP_n, in this case we can suppose that the vectors $\mathbf{x}_i$'s are the only nonzero binary vectors inside $\mathcal{L}_{\mathbf{x}}$.

Recall that additional binary vectors belong to $\bar{\mathcal{L}}_{\mathbf{x}}$, as discussed in Section 3.2.1 and 3.2.2. Indeed, the associated matrix $\mathbf{X}$ has columns of Hamming weight κ. So, denoting by $\mathbf{1}_s$ the vector of dimension s having all components equal to 1, this implies that

$$\mathbf{1}_n \cdot \mathbf{X} = \kappa \cdot \mathbf{1}_m$$

Therefore, we have that $\mathbf{1}_m \in \bar{\mathcal{L}}_{\mathbf{x}}$ and consequently $\mathbf{y}_i = \mathbf{1}_m - \mathbf{x}_i \in \bar{\mathcal{L}}_{\mathbf{x}}$ for any $i = 1, \ldots, n$, *i.e.* the completion of the lattice $\mathcal{L}_{\mathbf{x}}$ contains $n + 1$ new binary vector.

The linear relation between any $\mathbf{x}_i$ and $\mathbf{y}_i$ induces a linear relation among pairs of solutions. Indeed, if $\mathbf{C}$ is a basis of $\bar{\mathcal{L}}_{\mathbf{x}}$ got as the output of the orthogonal lattice attack, there exist vectors $\mathbf{e}, \mathbf{w}_i \in \mathbb{Z}^n$ such that $\mathbf{w}_i \cdot \mathbf{C} = \mathbf{x}_i$ for $i = 1, \ldots, n$ and $\mathbf{e} \cdot \mathbf{C} = \mathbf{1}_m$. Then $(\mathbf{e} - \mathbf{w}_i) \cdot \mathbf{C} = \mathbf{y}_i$. This implies that the vectors $\mathbf{f}_i = \mathbf{e} - \mathbf{w}_i$ are solutions of the multivariate quadratic system, too. If the vectors $\mathbf{x}_i$'s,$\mathbf{1}_m$ and $\mathbf{y}_i$'s are the unique binary vectors of $\bar{\mathcal{L}}_{\mathbf{x}}$, then $\mathbf{0}, \mathbf{e}, \mathbf{w}_1, \ldots \mathbf{w}_n, \mathbf{f}_1, \ldots \mathbf{f}_n$ are the only solutions of the multivariate system.

We consider then the following extended set

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

Because of the quadratic monomials, the dimension of $\mathrm{Span}(\mathcal{E})$ is larger[3] than n. As a consequence, Heuristic 1 is not suitable for a system (4.1) associated to random instances of HSSP_n^κ. Nevertheless, we can use the following heuristic:

Heuristic 2. *If $m > (n^2 + n)/2$, the dimension of* $\ker \mathbf{E}$ *is $2n$.*

We observed that Heuristic 2 is always satisfied in practice if the problem is not highly unbalanced; see Table 4.2.1, 4.2.2 and 4.2.3 for examples. However, this implies that we can not apply the eigenspace technique illustrated for HSSP_n. Indeed, we can only obtain a base matrix $\mathbf{K} \in \mathbb{Q}^{2n \times \frac{n^2+3n}{2}}$ of $\ker \mathbf{E}$ having form

$$\mathbf{K} = \begin{bmatrix} \mathbf{M} & \mathbf{I}_n \\ & \mathbf{0} \end{bmatrix}$$

where $\mathbf{M} \in \mathbb{Q}^{2n \times \frac{n^2+n}{2}}$. So, in this case we proceed by using a guessing technique, taking advantage of the symmetry among the solutions.

Recall, the last n components in the vectors in $\mathcal{Z}$ correspond to the linear part in the quadratic equations of (4.1). A vector $\mathbf{v} = (v_1, \ldots, v_n) \in \mathbb{Z}^n$ is then a solution of (4.3) if and only if there exists $v_{n+1}, \ldots, v_{2n} \in \mathbb{Z}$ such that $\mathbf{v}_+ = (v_1, \ldots, v_n, v_{n+1}, \ldots, v_{2n}) \cdot \mathbf{K} \in \mathcal{E}$. This gives:

$$(v_1, \ldots, v_n, v_{n+1}, \ldots, v_{2n}) \cdot \mathbf{M} = (v_i v_k)_{1 \le i \le k \le n} \tag{4.5}$$

Similarly to the previous case, *i.e.* for HSSP_n, we can also consider the problem reduced modulo a small prime p. Indeed, there always exists a p preserving the mentioned properties of the system. Therefore, in the following we directly describe the strategy implicitly assuming to be over a suitable $\mathbb{F}_p$.

Let us denote by $\mathbf{w}$ a generic $\mathbf{w}_i$ and $\mathbf{f}$ the corresponding $\mathbf{f}_i$, and $\mathbf{w} \cdot \mathbf{C} = \mathbf{x}$ and $\mathbf{f} \cdot \mathbf{C} = \mathbf{y}$. Recall also that any coordinate of $\mathbf{x}$ corresponds to a linear equation, *e.g.* $\langle \mathbf{w}, \tilde{\mathbf{c}}_1 \rangle = x_1$. For instance, then guessing the first coordinate of $\mathbf{w} \cdot \mathbf{C}$ provides directly n linearly independent quadratic equations on $w_j \langle \mathbf{w}, \tilde{\mathbf{c}}_1 \rangle - w_j x_1 = 0$ for $j = 1, \ldots, n$.

[3] See Section 4.2.3.

Such quadratic equations can be written as linear equations on the vector $\mathbf{v}_+$, according to (4.5), and used to split the space of solutions. Namely, guessing the first component determines two subspaces $V_{1,0}$ and $V_{1,1}$ of $\mathbb{F}_p^{2n}$, that corresponds to the conditions $x_1 = 0$ and $x_1 = 1$, respectively.

Since $\langle \mathbf{f}, \tilde{\mathbf{c}}_1 \rangle = 1 - x_1$, intersecting $\ker \mathbf{E}$ with $V_{1,0}$ and $V_{1,1}$ splits the space in about half. Indeed, $\ker \mathbf{E} \cap V_{1,0}$ and $\ker \mathbf{E} \cap V_{1,1}$ have dimension n and $n+1$, respectively. Clearly, the same argument applies to any coordinate of $\mathbf{x}$. Hence, for any $j = 1, \cdots, m$ the intersection with either $V_{j,0}$ or $V_{j,1}$ halves the space.

The cost of naively computing all the possible intersections requires performing an exponential number of operations. Notice that we had a similar issue for the eigenspaces intersection method previously discussed. Therefore, we can apply a branching strategy and split the space by iterating on coordinates j's that are pivots of $\mathbf{C}$. Without loss of generality, we suppose here that $\tilde{\mathbf{c}}_1, \ldots, \tilde{\mathbf{c}}_n$ are linearly independent modulo p. Moreover, we can directly start to iterate over the half space $\ker \mathbf{E} \cap V_{1,0}$, since the branching tree is symmetric.

Specifically, we start by computing $K_0 = \ker \mathbf{E} \cap V_{1,0}$, *i.e.* the preimage of $x_1 = 0$. This space can be further divided by intersecting it both with $V_{2,0}$ or $V_{2,1}$; as a result, we get two subspace $K_{00} = K_0 \cap V_{2,0}$ and $K_{01} = K_0 \cap V_{2,1}$, respectively. Obviously, we have $K_{0b} = \ker \mathbf{E} \cap V_{1,0} \cap V_{2,b}$. So, as a convention we write for $\mathbf{b} \in \{0,1\}^k$

$$K_{\mathbf{b}} = \ker \mathbf{E} \bigcap_{i \in \{1,\ldots,k\}} V_{i,b_i}$$

This branching strategy naturally creates an unbalanced binary tree, whose leaves are vector spaces (of dimension one) generated by each one of the coordinates' vectors of the elements of $\mathcal{E}$ with respect to the basis $\mathbf{K}$. Recall that as a root we actually consider K_0 rather than the full space, since if $\mathbf{x}_i$ (resp. $\mathbf{y}_i$) is retrieved starting by K_0, the dual $\mathbf{y}_i$ (resp. $\mathbf{x}_i$) can be computed directly using the linear relation. This implies that computing all the vectors among the $\mathbf{x}_i$'s and $\mathbf{y}_i$'s whose first coordinate is zero is sufficient. The full strategy is summarised by Algorithm 4.5.

We now discuss two possible situations, depending on the ratio between n and κ. For instance, suppose $\kappa/n = 1/2$. The fixed Hamming weight columns of $\mathbf{X}$ are independent, then we expect that at each level the dimension of $K_{\mathbf{b}}$ is halved. Moreover, we know the number of leaves is n. This implies that for a random instance of $\mathrm{HSSP}_n^{n/2}$ we can expect that the depth of the tree is $H = \mathcal{O}(\log n)$. See Figure 4.7.

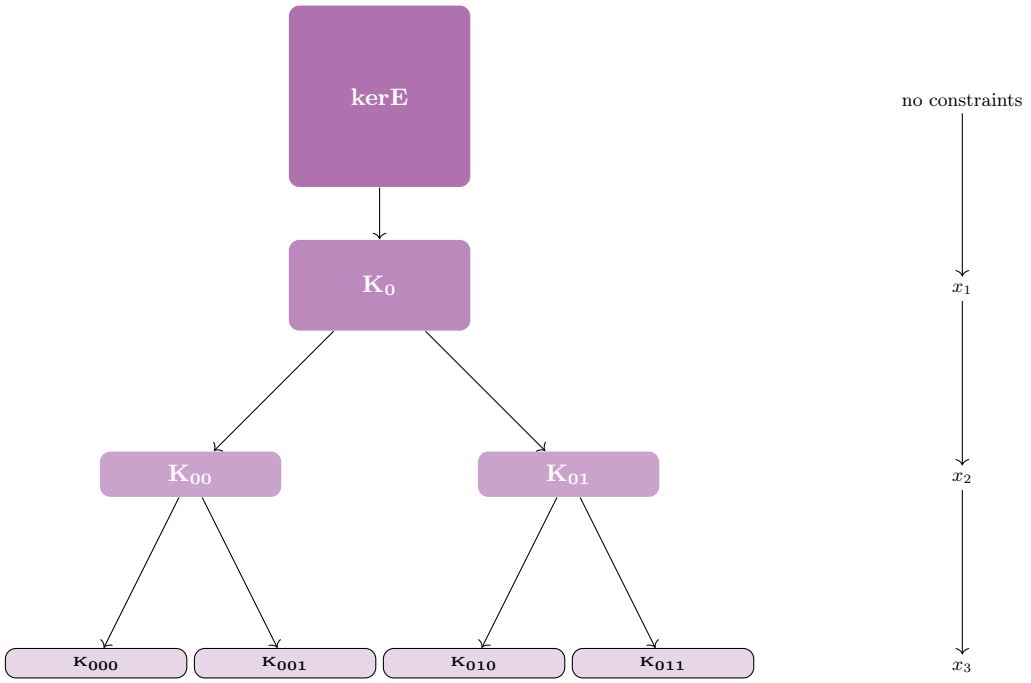


Fig. 4.7. Expected tree of the multivariate attack applied to HSSP_n^κ for $\kappa/n = 1/2$.

Instead, the shape of the tree is slightly different if the problem is not balanced. Namely, suppose that κ/n is a value neither extreme nor central, *e.g.* $\kappa/n = 1/4$; the probability of $x_i = 0$ is κ/n and the probability of $x_i = 1$ is $1 - \kappa/n$. This implies that we can expect the dimensions of the $K_{\mathbf{b}}$ not being equal, except for K_0 and K_1,. For instance, when $\kappa/n = 3/4$ we expect $\dim K_{\mathbf{b}} / \dim K_{\mathbf{b}0} \approx 4$ and $\dim K_{\mathbf{b}} / \dim K_{\mathbf{b}1} \approx 4/3$. See Figure 4.8.

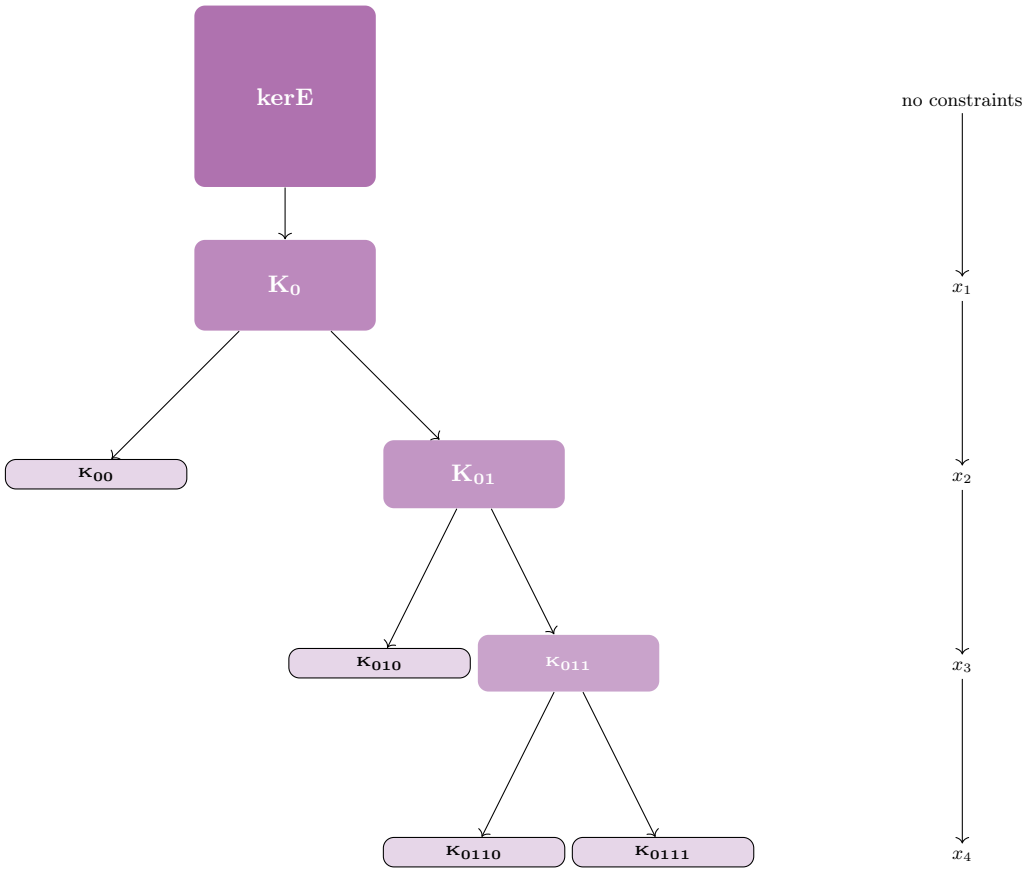


Fig. 4.8. Expected tree of the multivariate attack applied to HSSP_n^κ for $\kappa/n = 3/4$.

Therefore, if the problem is not perfectly balanced, the average height of the tree increases. Specifically, we formulate the following heuristic:

Heuristic 3. *For a random instance of* HSSP_n^κ *the height is* $H = \mathcal{O}(\log_a n)$*, where* $a = \min\{n/\kappa, n/(n-\kappa)\}$

Table 4.2.1, 4.2.2 and 4.2.3 collect experimental values of H for various sets of parameters.

Remark 4.9. Our bit-guessing method illustrated above produces the full list of binary vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n$ and $\mathbf{y}_1, \ldots, \mathbf{y}_n$. Since $\mathbb{E}[\|\mathbf{x}_i\|^2] = m\kappa/n$ and $\mathbb{E}[\|\mathbf{y}_i\|^2] = m(n-\kappa)/n$, it is not straightforward to distinguish any $\mathbf{x}_i$ from any $\mathbf{y}_i$ if the problem is balanced. In order to fully identify a target hidden basis, we need therefore a further step. Our algorithm for solving this final issue is described in Section 4.3.

Algorithm 4.5 Multivariate attack for HSSP_n^κ

Input: $\mathbf{C} \in \mathbb{Z}^{n\times m}$ a basis of $\bar{\mathcal{L}}_{\mathbf{x}}$.
Output: $\mathbf{z}_1, \ldots, \mathbf{z}_n \in \{0,1\}^m$, such that either $\mathbf{z}_i = \mathbf{x}_i$ or $\mathbf{z}_i = \mathbf{y}_i$ for $i = 1, \ldots, n$.

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 $1 \le j \le m$.
$\mathbf{E} = \left[\begin{array}{c} \mathbf{r}_1 \cdots \mathbf{r}_m \\ \hline -\mathbf{C} \end{array}\right] \in \mathbb{Z}^{\frac{n^2+3n}{2}\times m}$
$\mathbf{K} \leftarrow \ker_{\mathbb{F}_p} \mathbf{E}$ with $\mathbf{K} = \begin{bmatrix} \mathbf{M} & \mathbf{I}_n \\ \mathbf{0} & \end{bmatrix} \in {\mathbb{F}_p}^{2n\times\frac{n^2+3n}{2}}$
Write $\mathbf{M} = [\tilde{\mathbf{m}}_{ik}]_{1\le i\le k\le n}$ where $\tilde{\mathbf{m}}_{ik} \in {\mathbb{F}_p}^{2n}$.
Let $\mathbf{M}_i \in {\mathbb{F}_p}^{2n\times n}$ with $\mathbf{M}_i = [\tilde{\mathbf{m}}_{ik}]_{1\le k\le n}$, using $\tilde{\mathbf{m}}_{ik} := \tilde{\mathbf{m}}_{ki}$ for $i > k$.
$L \leftarrow [\mathbf{I}_{2n}]$
$C = True$
$j = 0$
while C **do**
    for $i \leftarrow 1$ to n **do**
        $\mathbf{g}_i \leftarrow \mathbf{M}_i \cdot \tilde{\mathbf{c}}_j$
    end for
    $\mathbf{G} = [\mathbf{g}_1, \ldots, \mathbf{g}_n] \in \mathbb{F}_p^{2n\times n}$
    $L_2 \leftarrow []$
    $\mathbf{V}_0 \leftarrow \ker \mathbf{G}$ ▷ This is a basis of $V_{j,0}$
    if $j > 0$ **then**
        $\mathbf{E}_1 \leftarrow \mathbf{G} - [\mathbf{I}_n|\mathbf{0}_n]^\intercal \in \mathbb{F}_p^{2n\times n}$
        $\mathbf{V}_1 \leftarrow \ker \mathbf{E}_1 \in \mathbb{F}_p^{2n\times n}$ ▷ This is a basis of $V_{j,1}$
    end if
    for all $\mathbf{V} \in L$ **do**
        if $\operatorname{rank} \mathbf{V} = 1$ **then**
            Append a generator $\mathbf{v}$ of $\mathbf{V}$ to L_2.
        else
            Compute a base matrix of $\mathbf{V} \cap \mathbf{V}_0$ and append it to L_2 if non-zero.
            if $j > 0$ **then**
                Compute a base matrix of $\mathbf{V} \cap \mathbf{V}_1$ and append it to L_2 if non-zero.
            end if
        end if
    end for
    $L \leftarrow L_2$
    $j \leftarrow j + 1$
    if $\max_{\mathbf{V}\in L} \operatorname{rank} \mathbf{V} = 1$ **then**
        $C \leftarrow False$
    end if
end while
$X \leftarrow []$
for all $\mathbf{v} \in L$ **do**
    Find $c \neq 0$ such that $\mathbf{x} = c \cdot \mathbf{v} \cdot \mathbf{C} \in \{0,1\}^m$, and append $\mathbf{x}$ to X.
end for
return X

Analysis

For each pair (j, b) computing the spaces $V_{j,b}$ requires $\mathcal{O}(n^2)$ operations. Those spaces are represented by matrices of given dimension, either $n \times 2n$ or $(n+1) \times 2n$, depending on b. Since any $K_{\mathbf{b}}$ has dimension at most $n \times 2n$ this implies that the computation of each node is $\mathcal{O}(n^3)$. Then the full bit-guessing algorithm performs $\mathcal{O}(n^3 \cdot 2^H)$ arithmetic operations; thus, heuristically this is $\mathcal{O}(n^3 \cdot 2^{\log_a n}) = \mathcal{O}(n^4)$.

Therefore, the overall complexity of the multivariate approach is dominated by the cost of computing the kernel of the matrix $\mathbf{E}$, as for the eigenspace approach previously described for HSSP_n. Then, if Heuristic 2 and Heuristic 3 hold true, we can recover the binary vectors within time complexity $\mathcal{O}(n^6)$ and space complexity $\mathcal{O}(n^4)$. Experimental results are collected in Section 4.4.2.

Experiments on Heuristic 3

Table 4.2.1, 4.2.2 and 4.2.3 collect experimental values of H, as in Heuristic 3, for various set of pairs n, κ and $m = n(n+4)/2$. We remark that Heuristic 2 was always verified for these choices of parameters in our experiments.

n	κ	a	$\log_a(n)+1$	$\hat{\mathbb{E}}[H]$
90	10	1.12	39.20	37.60
90	20	1.29	18.91	20.90
90	30	1.50	12.10	14.60
90	40	1.80	8.66	14.30
90	45	2.00	7.49	14.80
90	50	1.80	8.66	14.30
90	55	1.64	10.14	13.90
90	60	1.50	12.10	15.70
90	70	1.29	18.91	21.40
90	80	1.12	39.20	41.50

Table 4.2.1. Collection of $a = \min\{n/\kappa, n/(n-\kappa)\}$, $\log_a(n)+1$ and experimental mean of H over 10 random instances of $\mathrm{HSSP}^{\kappa}_{90}$, for each κ. Heuristic 2 was always verified for these choices of n and κ.

4.2.3 Dimension of $\ker \mathbf{E}$ and number of solutions

Consider the set of vectors associated to the coordinates of the binary vectors $\mathbf{x}_i$'s respect to $\mathbf{C}$:

n	κ	a	$\log_a(n)+1$	$\hat{\mathbb{E}}[H]$
110	10	1.10	50.32	55.20
110	20	1.22	24.42	29.40
110	30	1.38	15.76	20.60
110	40	1.57	11.40	15.00
110	50	1.83	8.75	13.90
110	55	2.00	7.78	15.40
110	60	1.83	8.75	15.10
110	70	1.57	11.40	16.70
110	80	1.38	15.76	18.60
110	90	1.22	24.42	26.00
110	100	1.10	50.32	48.00

Table 4.2.2. Collection of $a = \min\{n/\kappa, n/(n-\kappa)\}$, $\log_a(n)+1$ and experimental mean of H over 10 random instances of $\mathrm{HSSP}^{\kappa}_{110}$, for each κ. Heuristic 2 was always verified for these choices of n and κ.

n	κ	a	$\log_a(n)+1$	$\hat{\mathbb{E}}[H]$
130	20	1.18	30.14	28.00
130	40	1.44	14.24	19.00
130	50	1.62	11.03	16.33
130	65	2.00	8.02	15.00
130	90	1.44	14.24	18.50
130	110	1.18	30.14	32.00

Table 4.2.3. Collection of $a = \min\{n/\kappa, n/(n-\kappa)\}$, $\log_a(n)+1$ and experimental mean of H, computed over 3 random instances of $\mathrm{HSSP}^{\kappa}_{130}$ for each κ. Heuristic 2 was always verified for these pairs of n and κ.

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

Obviously, $\{\mathbf{0}\} \cup \mathcal{W} \subseteq \mathcal{Z} \cap \ker \mathbf{E}$. Since by assumption the vectors $\mathbf{w}_i$'s are linearly independent, $\mathrm{Span}(\mathcal{W})$ is a subspace of dimension n of $\ker \mathbf{E}$. This implies that $\dim \ker \mathbf{E} \geq n$, and consequently $n \leq \mathrm{rank}\, \mathbf{E} \leq \min\left\{m, \frac{n^2+n}{2}\right\}$.

In Section 4.2 [CG20a] it is conjectured that for $m > (n^2+n)/2$, the matrix $\mathbf{R}$ is likely to be of rank $(n^2+n)/2$ when generated by a random instance of HSSP_n Namely, for such values of m we can expect $\dim \ker \mathbf{E} = n$, and that a basis of $\ker \mathbf{E}$ is given by the set $\mathcal{W}$. This is in fact always verified in practice. However, we observed that this is no longer true when considering a random instance of HSSP^{κ}_n. The main reason is that the correct intuition is actually that the dimension of $\ker \mathbf{E}$ depends on the number of solutions. Since we showed

in Section 3.2.2 that additional binary vectors are in $\bar{\mathcal{L}}_{\mathbf{x}}$ in such a case, Heuristic 1 cannot be correct for HSSP_n^κ.

Providing a precise and formal explanation of our new claim would require introducing some background, without eventually changing the *heuristic* status of the algorithm. Therefore, the purpose of this section is just to give an informal intuition of a possible reasoning behind our conjecture.

System (4.1) is defined by the quadratic multivariate polynomials $\langle \mathbf{w}, \tilde{\mathbf{c}}_j \rangle^2 - \langle \mathbf{w}, \tilde{\mathbf{c}}_j \rangle$ for $j = 1, \ldots, m$. Studying the system algebraically is equivalent to considering the ideal $I \subseteq \mathbb{Q}[\mathbf{w}]$ generated by such polynomials. Now, let W_k be the span of the monomial of degree up to k and $I_k = I \cap W_k$. For instance, consider

$$W_2 = \langle \{w_i w_k\}_{1 \leq i \leq k \leq n}, \{w_i\}_{1 \leq i \leq n}, 1 \rangle_{\mathbb{Q}}$$

and $I_2 = V_2 \cap I$. By construction, the rows of $\mathbf{E}$ are the coordinates of the polynomials $\langle \mathbf{w}, \tilde{\mathbf{c}}_j \rangle^2 - \langle \mathbf{w}, \tilde{\mathbf{c}}_j \rangle$ for $j = 1, \ldots, m$ respect to the aforementioned monomial basis of W_2. Therefore, we have that $\dim_{\mathbb{Q}} I_2 \geq \operatorname{rank} \mathbf{E}$, and consequently $\dim_{\mathbb{Q}} W_2/I_2 \leq \dim \ker \mathbf{E} + 1$.

Thus, we now observe that $\dim_{\mathbb{Q}} \mathbb{Q}[\mathbf{w}]/I$ is linked to the number of solutions. Indeed, if $\mathbf{q}_0 \in \mathcal{V}_{\mathbb{C}}(I)$, *i.e.* $\mathbf{q}_0 \in \mathbb{C}^n$ is a solution of (4.1), then $\mathbf{q}_0 \cdot \mathbf{C} \in \{0,1\}^m$. This implies that $\mathcal{V}_{\mathbb{C}}(I)$ has finite cardinality, since $\mathbf{C}$ is full rank. It can be proven that this implies that ideal I has dimension zero and so the dimension of $\mathbb{Q}[\mathbf{w}]/I$ as $\mathbb{Q}$ vector space is finite. This is convenient because zero dimensional ideals have good properties. For instance, it is possible to prove that

$$\dim_{\mathbb{Q}} \mathbb{Q}[\mathbf{w}]/I = \dim_{\mathbb{C}} \mathbb{C}[\mathbf{w}]/(I\mathbb{C}[\mathbf{w}]) \geq \#\mathcal{V}_{\mathbb{C}}(I),$$

where the last inequality becomes an equality if I is radical. Since $\dim_{\mathbb{Q}} \mathbb{Q}[\mathbf{w}]/I = \dim_{\mathbb{Q}} V_2/I_2 + \Xi$, the previous inequalities imply that

$$\dim \ker \mathbf{E} \geq \mathcal{V}_{\mathbb{C}}(I) - 1 - \Xi \geq \mathcal{V}_{\mathbb{Z}}(I) - 1 - \Xi$$

Then, the number of solutions (together with other properties of the ideal associated to the system) affects the dimension of $\dim \ker \mathbf{E}$. A similar (but not exactly the same) reasoning can be applied for working over $\mathbb{F}_p$.

After defining the linearised system (4.3), we claimed that our method relies on the heuristic assumption that *the matrix* $\mathbf{E}$ *has the maximal "possible" rank when* $m > (n^2 + n)/2$. In view of the given argument, therefore, we can interpret this as assuming that the previous conditions are as tight as possible. Obviously, the foregoing does not actually properly justify our heuristics; however, it partially explains why we can expect the two families of problems to have different behaviours.

4.3 Retrieving fixed Hamming weight bases

The methods we discussed for solving the hidden subset sum problem require dealing with several sub-problems, such as disclosing a hidden lattice and solving a multivariate polynomial system. We have seen that for random instances of HSSP_n our algorithms directly reveals the hidden binary vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n$. However, for balanced HSSP_n^κ this is not the case. Indeed, we have shown that the lattice $\bar{\mathcal{L}}_{\mathbf{x}}$ contains additional binary vectors $\mathbf{y}_i = \mathbf{1}_m - \mathbf{x}_i$ for any $i = 1, \ldots, n$, since the matrix $\mathbf{X}$ has columns of fixed Hamming weight κ. When $\kappa/n \approx 1/2$, it is almost impossible to directly distinguish between any $\mathbf{y}_i$ and $\mathbf{x}_i$. Nevertheless, in this section we present an algorithm that finally retrieves the target vectors $\mathbf{x}_i$'s.

Before introducing our new algorithm, it is useful to notice that we can easily distinguish the (unordered) pairs $\{\mathbf{y}_i, \mathbf{x}_i\}$. In fact, given the set of vectors $\{\mathbf{x}_1, \ldots, \mathbf{x}_n, \mathbf{y}_1, \ldots, \mathbf{y}_n\}$, we can take those ones having first component equal to zero and get the dual vector by subtraction from $\mathbf{1}_m$.[4]

Thus, the remaining task consists in deciding for each pair which vector is part of the basis. Enumerating all the possible bases requires to produce 2^n matrices, which is clearly not efficient; therefore, in the first part of this section we describe an iterative algorithm for completing this task in polynomial time.

Notice that this algorithm cannot always be integrated with the BKZ attack of Section 3.1.2. Indeed, as our analysis in Section 3.2.2 highlighted, the vector $\mathbf{1}_m$ is likely to appear in the BKZ reduced basis, which implies that one of the pairs $\{\mathbf{y}_i, \mathbf{x}_i\}$ remains unknown. Nevertheless, in Section 4.3.2 we present a variant of our iterative algorithm that recovers fixed Hamming weight bases, even if one pair is missing.

4.3.1 Our algorithm

Consider the set of unordered pairs $\{\{\mathbf{y}_i, \mathbf{x}_i\}\}_{i=1,\ldots,n}$. Then for each $i = 1 \ldots, n$ we randomly select a row vector $\mathbf{z}_i$ from the ith pair, *i.e.* $\mathbf{z}_i$ is either equal to $\mathbf{y}_i$ or the corresponding $\mathbf{x}_i$. The input of our iterative algorithm is a matrix $\mathbf{Z}$ whose rows are $\mathbf{z}_i$'s produced by this procedure.

We start by measuring the variance of the vector $\mathbf{s} = \mathbf{1}_n \cdot \mathbf{Z}$. Namely, we compute the sample mean $\hat{\mu}_s = \sum_{i=1}^m s_i/m$ and the sample variance $v_0 = \sum_{i=1}^m (s_i^2 - \hat{\mu}_s^2)/m$. Subsequently, we iteratively *twist* the rows of $\mathbf{Z}$, by exchanging $\mathbf{z}_i$ with $\mathbf{1}_m - \mathbf{z}_i$, keeping the twisted vector if and only if the variance of the new $\mathbf{s}$ decreases. The output of the algorithm is either the matrix $\mathbf{Y}$ of all $\mathbf{y}_i$'s or $\mathbf{X}$ that one of all $\mathbf{x}_i$'s. Our iterative algorithm is illustrated by Algorithm 4.6.

Finally, to distinguish $\mathbf{X}$ and $\mathbf{Y}$ is sufficient to solve the linear system for computing the vector $\boldsymbol{\alpha}$ of the hidden weight and if the chosen basis is correct, then the problem is eventually solved. Obviously, given one of the two matrix computing the other one is straightforward.

[4] Notice that this is straightforward if taking the output of the bit-guessing algorithm of Section 4.2.2.

Algorithm 4.6 Statistical attack for retrieving fixed Hamming weight bases

Input: $\mathbf{Z} \in \{0,1\}^{n \times m}$ whose rows are either one of the $\mathbf{y}_i$'s or $\mathbf{x}_i$'s, *e.g.* the output of Algorithm 4.5.

Output: $\mathbf{Z} \in \{0,1\}^{n \times m}$ whose columns have the same Hamming weight.

Let $\mathbf{z}_j$ for $1 \leq j \leq n$ be the rows of $\mathbf{Z}$.
$\mathbf{s} = \mathbf{1}_n \cdot \mathbf{Z}$
$\hat{\mu}_s = \sum_{i=1}^{m} s_i/m$
$v_0 = \sum_{i=1}^{m} (s_i^2 - \hat{\mu}_s^2)/m$
$v = v_0$
$i = 0$
while $v \neq 0$ **do**
    $j \leftarrow i \mod n$
    $\mathbf{z}_j \leftarrow \mathbf{1}_m - \mathbf{z}_j$
    $\mathbf{s} \leftarrow \mathbf{1}_n \cdot \mathbf{Z}$
    $\hat{\mu}_s \leftarrow \sum_{i=1}^{m} s_i/m$
    $v \leftarrow \sum_{i=1}^{m} (s_i^2 - \hat{\mu}_s^2)/m$
    if $v < v_0$ **then**
        $v_0 \leftarrow v$
    else
        $\mathbf{z}_j \leftarrow \mathbf{1}_m - \mathbf{z}_j$
    end if
    $i \leftarrow i + 1$
end while
return Z

Analysis of Algorithm 4.6

Algorithm 4.6 exploits two properties of the columns of $\mathbf{X}$: they are sampled independently and they have fixed Hamming weight. In this section we provide a formal analysis of our algorithm, proving both the correctness and convergence.

Let $\boldsymbol{\epsilon} = (\epsilon_1, \ldots, \epsilon_n)$ be a random vector of dimension n uniformly distributed over $\mathcal{B}_{n,\kappa}$, we define $s_\kappa = \langle \boldsymbol{\epsilon}, \mathbf{1}_n \rangle$. s_κ is a constant random variable. Namely, $\mathbb{E}[s_\kappa] = \kappa$ and $\sigma^2_{s_\kappa} = 0$. By Definition 2.2, the columns of $\mathbf{X}$ are sampled from the uniform distribution over $\mathcal{B}_{n,\kappa}$; therefore, we can interpret them as a samples of $\boldsymbol{\epsilon}$. In addition, we have that $\hat{\mathbb{E}}[s_\kappa] = \mathbf{1}_n \cdot \mathbf{X}/m = \kappa$ and $\hat{\sigma}^2_{s_\kappa} = 0$.

Recall that as input of the algorithm we have n row vectors, but for each one we do not know if we actually have either one the of $\mathbf{x}_i$'s or the corresponding $\mathbf{y}_i$. Thus, we now introduce a further kind of random variable in order to formalise this fact. Namely, for any $i = 1, \ldots, n$, we want to define a random vector whose coordinate can be either ϵ_i or $1 - \epsilon_i$ with the same probability. Then, we the define the following family of random vectors parameterised by binary vectors of dimension n: for any $\mathbf{b} = (b_1, \ldots, b_n) \in \mathcal{B}_n$, $\boldsymbol{\rho}(\mathbf{b})$ is the random vector whose ith coordinate is

$$b_i\epsilon_i + (1-b_i)(1-\epsilon_i) = 1 - b_i + (2b_i - 1)\epsilon_i$$

any $i = 1, \ldots, n$.

If $\mathbf{Z}$ is any matrix such that any row $\mathbf{z}_i$ is sampled uniformly from $\{\mathbf{y}_i, \mathbf{x}_i\}$, then each column of $\mathbf{Z}$ can be interpreted as randomly sampled from the distribution of $\boldsymbol{\rho}(\mathbf{b})$ for a fixed unknown $\mathbf{b}$. We can associate to each loop of Algorithm 4.6 a fixed $\mathbf{b}$, and so twisting the ith variable corresponds to twisting the ith bit of $\mathbf{b}$.

Let us consider now a single loop, *i.e.* $\mathbf{b}$ is fixed. Writing $\boldsymbol{\rho}(\mathbf{b}) = \boldsymbol{\rho} = (\rho_1, \ldots, \rho_n)$ we define

$$s = \sum_{i=1}^{n} \rho_i.$$

We have then that $\mathbb{P}(s = s_\kappa) = 2^{-n}$ and $\mathbf{s} = \mathbf{1}_n \cdot \mathbf{Z}$ is a vector of samples distributed according to s. Therefore, if m is sufficiently large, we can get a good estimation of σ_s^2.

We are going two show that the value σ_s^2 can be used to measure (informally) the "distance" of $\boldsymbol{\rho}$ from either $\boldsymbol{\epsilon}$ or $1 - \boldsymbol{\epsilon}$. Recall that these two cases correspond to two target settings: we have $\mathbf{z}_i = \mathbf{x}_i$ for every $i = 1, \ldots, n$ if $\boldsymbol{\rho} = \boldsymbol{\epsilon}$; while $\mathbf{z}_i = \mathbf{y}_i$ for every $i = 1, \ldots, n$ if $\boldsymbol{\rho} = \mathbf{1} - \boldsymbol{\epsilon}$.

For each $1 \leq i, j \leq n$ and $i \neq j$ we define

$$\Delta_{i,j}^{\mathbf{b}} = \begin{cases} 1 & (\rho_i = \epsilon_i) \wedge (\rho_j = \epsilon_j) \text{ or } (\rho_i = 1 - \epsilon_i) \wedge (\rho_j = 1 - \epsilon_j) \\ 0 & \text{otherwhise.} \end{cases}$$

saying that ρ_i and ρ_j are *compatible* if $\Delta_{i,j}^{\mathbf{b}} = 1$. Then if $\Delta_i^{\mathbf{b}} = \sum_{i \neq j} \Delta_{i,j}^{\mathbf{b}} + 1$, we call *degree of compatibility* of $\boldsymbol{\rho}$

$$\Delta^{\mathbf{b}} = \max_{1 \leq i \leq n} \Delta_i^{\mathbf{b}}.$$

i.e. the max number of compatible pairs of variables in $\boldsymbol{\rho}$. For instance, $\boldsymbol{\epsilon}$ and $1 - \boldsymbol{\epsilon}$ have the maximal possible degree of compatibility n. Conversely, the minimal degree of compatibility is either $n/2 + 1$ or $n/2$, if n is odd or even, respectively.

In the following we show the relation between σ_s^2 and the degree of compatibility of $\boldsymbol{\rho}$, and we prove that by twisting the rows of $\mathbf{Z}$ we can decrease σ_s^2 and consequently maximise the degree of compatibility of $\boldsymbol{\rho}$. Specifically, we prove that the variance σ_s^2 can increase only if, by twisting ρ_j, we decrease the degree of compatibility of $\boldsymbol{\rho}$. This is a direct consequence of the following lemma:

Lemma 4.10. *Let* $\mathbf{b} = (b_1, \ldots, b_n) \in \mathcal{B}_n$ *and* $j \in \{1, \ldots, n\}$. *Denoting:*

- $\mathbf{b}' \in \mathcal{B}_n$ *such that* $b'_i = b_i$ *for* $i \neq j$ *and* $b'_j = 1 - b_j$,
- $\boldsymbol{\rho}(\mathbf{b}) = \boldsymbol{\rho}$ *and* $\boldsymbol{\rho}(\mathbf{b}') = \boldsymbol{\rho}'$,
- $s' = 1 - \rho_j + \sum_{i \neq j} \rho_i = \sum_i \rho'_i$.

we have that $\sigma_{s'}^2 > \sigma_s^2$, *if and only if* $\Delta_j^{\mathbf{b}} > (n+1)/2$.

Proof. Recall that $c_{\rho_i, \rho_j} = \mathbb{E}[\rho_i \rho_j] - \mathbb{E}[\rho_i]\mathbb{E}[\rho_j]$ is the correlation of ρ_i and ρ_j; we have that

$$\sigma_s^2 = \sum_{i=1}^{n} \sigma_{\rho_i}^2 + 2 \sum_{1 \le i < j \le n} c_{\rho_i,\rho_j}$$

Since $\sigma_{\rho_i}^2 = \sigma_{1-\rho_i}^2 = \sigma_{\epsilon_i}^2 = \kappa(n-\kappa)/n^2$ and $\sigma_{s_\kappa}^2 = 0$ we have

$$\sigma_s^2 = \sigma_s^2 - \sigma_{s_\kappa}^2 = 2 \sum_{1 \le i < j \le n} (c_{\rho_i,\rho_j} - c_{\epsilon_i,\epsilon_j}) \tag{4.6}$$

In addition, notice that

$$c_{1-\rho_i,\rho_j} = \mathbb{E}[(1-\rho_i)\rho_j] - \mathbb{E}[1-\rho_i]\mathbb{E}[\rho_j] = \mathbb{E}[\rho_j] - \mathbb{E}[\rho_i\rho_j] - \mathbb{E}[1]\mathbb{E}[\rho_j] + \mathbb{E}[\rho_i]\mathbb{E}[\rho_j] = -c_{\rho_i,\rho_j}$$

and, since $c_{\epsilon_i,\epsilon_j} = -\kappa(n-\kappa)/(n^2(n-1))$, this implies

$$c_{\rho_i,\rho_j} = \begin{cases} -\frac{\kappa(n-\kappa)}{n^2(n-1)} & \text{if } \rho_i, \rho_j \text{ are compatible} \\ \frac{\kappa(n-\kappa)}{n^2(n-1)} & \text{otherwhise.} \end{cases} = (-1)^{\Delta_{i,j}} \frac{\kappa(n-\kappa)}{n^2(n-1)}$$

Without loss of generality, suppose we twist the first variable of $\boldsymbol{\rho}$ and we denote $s' = 1 - \rho_1 + \sum_{i=2}^{n} \rho_i$. Then we get

$$\begin{aligned} \sigma_{s'}^2 - \sigma_s^2 &= -4 \sum_{2 \le j \le n} c_{\rho_1,\rho_j} = -4 \frac{\kappa(n-\kappa)}{n^2(n-1)} \cdot \sum_{2 \le j \le n} (-1)^{\Delta_{1,j}} = \\ &= -4 \frac{\kappa(n-\kappa)}{n^2(n-1)} \cdot [-(\Delta_1 - 1) + (n - 1 - (\Delta_1 - 1))] = \\ &\quad -4 \frac{\kappa(n-\kappa)}{n^2(n-1)} \cdot (n - 2\Delta_1 + 1) \end{aligned}$$

This implies that if $\sigma_{s'}^2 > \sigma_s^2$, if and only if $\Delta_1 > (n+1)/2$, *i.e.* ρ_1 is actually compatible with largest part of the other variables. Namely, in such a case twisting ρ_1 we are actually decreasing the degree of compatibility of $\boldsymbol{\rho}$.

□

Therefore, we can conclude that, after twisting any of the ρ_i's, we can just compute the new value of σ_s^2 and keep the twist only if this value is lower than the previous. This is indeed the loop of Algorithm 4.6, since in terms of samples twisting ρ_i corresponds exactly to twisting $\mathbf{z}_i$.

This implies that the while loop of Algorithm 4.6 terminates with the correct answer, as eventually it will reach $\sigma_s^2 = 0$. The worst case scenario in terms of number of loops is when we have roughly half rows $\mathbf{x}_i$'s and the other half $\mathbf{y}_i$'. Nevertheless, this requires anyway less than $2n$ loops, which implies that the algorithm is also polynomial.

n	κ	#twist	prob. extra-twist
40	20	58.10	0.50
50	25	75.10	0.50
60	30	84.70	0.70
70	35	93.70	0.50
80	40	125.00	0.70
90	45	131.60	0.40
110	55	154.70	0.30

Table 4.3.1. Average number of twisting loops of Algorithm 4.6 applied to the output of the multivariate attack, over 10 random instances of $\mathrm{HSSP}_n^{n/2}$. The last column displays the ratio between the number of times that the $\mathbf{y}_i$ have been returned, hence a final round of twists have been necessary to retrieve the $\mathbf{x}_i$'s, and the total number of runs.

For large values of m, *e.g.* $m \approx n^2/2$ the sample variance is a sufficiently good estimation of the variance, and as previously observed, $\mathbf{s} = \mathbf{1}_n \cdot \mathbf{Z}$ is a vector of m independent samples distributed according to s. Hence, Algorithm 4.6 succeeds in finding either $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ or $\{\mathbf{y}_1, \ldots, \mathbf{y}_n\}$ in polynomial time.

Table 4.3.1 is a collection of experimental runs of the algorithm after applying the multivariate attack to a random instance of $\mathrm{HSSP}_n^{n/2}$. The practical experimental confirm our analysis and show that the probability of finding either $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ or $\{\mathbf{y}_1, \ldots, \mathbf{y}_n\}$ is about $1/2$ when $\kappa = n/2$.

4.3.2 Variant for the missing pair case

In this section we show how to adapt Algorithm 4.6 when one of the pairs is missing. Without loss of generality suppose that is $\{\mathbf{x}_1, \mathbf{y}_1\}$. The main idea is to use the following relation:

$$\kappa \mathbf{1}_m - \sum_{j=2}^{n} \mathbf{x}_j = \mathbf{x}_1 \tag{4.7}$$

When all the pairs are known, we showed that our algorithm converges to a variance zero vector. Instead, here we want to use the fact that we know the expected variance of the coordinates of $\mathbf{x}_1$ and $\mathbf{y}_1$, *i.e.* $\kappa(n-\kappa)/n^2$.

From (4.7), we have that

$$\mathbf{1}_m + \sum_{j=2}^{n} \mathbf{x}_j = (\kappa+1)\mathbf{1}_m - \mathbf{x}_1$$

Since $(\kappa+1)\mathbf{1}_m$ is a constant vector and opposite variables have the same variance, we obtain that the expected variance of the coordinates of $\mathbf{1}_m + \sum_{j=2}^{n} \mathbf{x}_j$ is $\kappa(n-\kappa)/n^2$. Similarly, we can derive that the expected variance of $\mathbf{1}_m + \sum_{j=2}^{n} \mathbf{y}_j$ is $\kappa(n-\kappa)/n^2$, too.

Therefore, we apply a strategy which is a variant of Algorithm 4.6. Namely, we start from a matrix $\mathbf{Z}$ whose first row is $\mathbf{1}_m$ and the other rows are only one among $\mathbf{y}_i$ and $\mathbf{x}_i$ for each $i = 2 \ldots, n$; we measure the variance of the vector $\mathbf{s}_0 = \mathbf{1}_n \cdot \mathbf{Z}$ and iteratively we *twist* the rows of $\mathbf{Z}$, by exchanging $\mathbf{z}_i$ with $\mathbf{1}_m - \mathbf{z}_i$, keeping the twisted vector if only if the variance of the new $\mathbf{s}_0$ decreases. Obviously, the twisting of the first row can be avoided.

This algorithm should converge to a vector with prescribed variance, *i.e.* either equal to $(\kappa + 1)\mathbf{x}_1$ or $(n - \kappa + 1)\mathbf{y}_1$. The fixed Hamming weight basis will be then either $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ or $\{\mathbf{y}_1, \ldots, \mathbf{y}_n\}$, respectively.

Our iterative algorithm is illustrated by Algorithm 4.7.

Algorithm 4.7 Statistical attack for retrieving fixed Hamming weight bases with a missing pair

Input: $\mathbf{Z} \in \{0,1\}^{n \times m}$ whose first row is $\mathbf{1}_m$ and the other rows are either one of the $\mathbf{y}_i$'s or $\mathbf{x}_i$'s, *e.g.* the output of Algorithm 4.5. ι_κ a tolerance level.
Output: $\mathbf{Z} \in \{0,1\}^{n \times m}$ whose columns have the same Hamming weight.

Let $\mathbf{z}_j$ for $1 \leq j \leq n$ be the rows of $\mathbf{Z}$.
$\mathbf{s} = \mathbf{1}_n \cdot \mathbf{Z}$
$\hat{\mu}_s = \sum_{i=1}^{m} s_i/m$
$v_0 = \sum_{i=1}^{m} (s_i^2 - \hat{\mu}_s^2)/m$
$v = v_0$
$v_\kappa = \kappa \cdot (n - \kappa)/n^2$
$i = 0$
while $|v_\kappa - v| > \iota_\kappa$ **do**
    $j \leftarrow i \mod n$
    if $j \neq 1$ **then**
        $\mathbf{z}_j \leftarrow \mathbf{1}_m - \mathbf{z}_j$
        $\mathbf{s} \leftarrow \mathbf{1}_n \cdot \mathbf{Z}$
        $\hat{\mu}_s \leftarrow \sum_{i=1}^{m} s_i/m$
        $v \leftarrow \sum_{i=1}^{m} (s_i^2 - \hat{\mu}_s^2)/m$
        if $v < v_0$ **then**
            $v_0 \leftarrow v$
        else
            $\mathbf{z}_j \leftarrow \mathbf{1}_m - \mathbf{z}_j$
        end if
    end if
    $i = i + 1$
end while
$\mathbf{z} = \sum_{i>1} \mathbf{z}_i$
$\kappa_0 = \max(\mathbf{z})$
$\mathbf{z}_1 \leftarrow (\kappa_0 \mathbf{1}_m - \mathbf{z})$
return Z

Analysis of Algorithm 4.7

We adopt for the analysis of Algorithm 4.7 the notation established in Section 4.3.1. Then, we define

$$t = 1 + \sum_{i=2}^{n} \rho_i.$$

and we have that

$$\sigma_t^2 = \sum_{i=2}^{n} \sigma_{\rho_i}^2 + 2 \sum_{2 \le i < j \le n} c_{\rho_i, \rho_j}.$$

Hence, using similar arguments we obtain the following variant of Lemma 4.10:

Lemma 4.11. *Let us denote* $t' = 1 + (1 - \rho_j) + \sum_{i \notin \{1,j\}} \rho_i$ *for some* $j \in \{2, \ldots, n\}$. *Then* $\sigma_{t'}^2 > \sigma_t^2$, *if and only if* $\Delta_j^{\mathfrak{b}} > (n+1)/2$.

This implies that the variance σ_t^2 can increase only if, by twisting ρ_j, we decrease the degree of compatibility of $\boldsymbol{\rho}$. So again, for large values of m, *e.g.* $m \approx n^2/2$ Algorithm 4.6 succeeds in finding either $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ or $\{\mathbf{y}_1, \ldots, \mathbf{y}_n\}$ in polynomial time. However, considering the purpose of including this algorithm after Nguyen-Stern attack, if the value of m considered is small, as $2n$, the probability that the sample estimation is precise enough actually decreases; this can lead both to slower convergence or divergence of the algorithm in certain cases.

4.3.3 Practical experiment for Nguyen-Stern with fixed Hamming weight

We observed that if $\varsigma = \kappa/n \approx 1/2$, it is difficult to distinguish the vectors $\mathbf{x}_i$'s from the $\mathbf{y}_i$'s when solving HSSP_n^{κ}. Therefore, we described our statistical algorithm for this purpose, and its variant working also when a pair is missing. However, this requires a slightly larger m, *e.g.* $m \ge 4n$. In Table 4.3.2 we present then the result of practical experiments of the Nguyen-Stern algorithm, *including* our new algorithm present in this section. We denote as *extra* the operations performed after BKZ, as for instance the algorithm to retrieve the fixed Hamming weight basis.

4.4 Practical experiments for our multivariate attack

We provide in this section the result of practical experiments with our algorithm based on a multivariate approach described in previous sections. Specifically, the algorithm is composed of two main steps: the improved orthogonal lattice attack of Section 4.1 and a binary vectors' search based on solving a quadratic multivariate polynomial system, illustrated in Section 4.2. 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

				Step 1		Step 2			
n	κ	m	$\log Q$	LLL $\mathcal{L}_0$	LLL $\mathcal{L}_{\mathbf{x}}^{\perp}$	Reduction		extra	Total
70	35	280	772	7.2 s	1.8 s	LLL	3.7 s	0.4 s	13.3 s
90	45	360	1151	18.3 s	6.2 s	BKZ-10	8.2 s	1.9 s	34.7 s
110	55	440	1592	43.3 s	11.3 s	BKZ-20	25.8 s	1.5 s	82.0 s
130	65	520	2095	101.1 s	31.7 s	BKZ-30	52.9 s	3.9 s	3 min
150	75	600	2659	4 min	57.8 s	BKZ-50	4 min	10.5 s	9 min
170	85	850	3282	17 min	98.2 s	BKZ-60	34 min	27.4 s	53 min
190	95	950	3965	21 min	154.9 s	BKZ-80	1510 min	8.6 s	25 h

Table 4.3.2. Running time of the Nguyen-Stern attack for $\mathrm{HSSP}_n^{n/2}$.

is much faster. For these size-reductions, we apply the technique described in Section 4.1.2 with parameter $k = 4$. As modulus bitsize we set $\log Q \simeq 2\iota n^2 + n \cdot \log n$ with $\iota = 0.035$; see Section 3.2.1 and 4.1. Furthermore, we recall that, as discussed in Section 4.2.2, for the second step we have to use two different strategies depending whether we are considering HSSP_n or HSSP_n^{κ}. Therefore, in the following we provide our experimental results for both cases. The content of this section partially appeared in [CG20a]. Our `SageMath` implementation is available at `https://github.com/agnesegini/solving_hssp`.

4.4.1 HSSP_n

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 4.4 (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 Section 4.2.1, these operations can be performed modulo 3. As illustrated in Table 4.4.1, computing the kernel is the most time consuming part of Step 2, while the computation of the eigenspaces (also modulo 3) to recover the original vectors $\mathbf{x}_i$ is quite fast.

			Step 1		Step 2			
n	m	$\log Q$	LLL $\mathcal{L}_0$	LLL $\mathcal{L}_{\mathbf{x}}^{\perp}$	Kernel mod 3	Eigenspaces	Total S2	Total
70	2590	772	9.5 s	2.1 s	20.1 s	0.9 s	25.2 s	37.0 s
90	4230	1151	29.0 s	5.7 s	46.1 s	0.3 s	57.7 s	92.8 s
110	6270	1592	73.0 s	20.5 s	87.1 s	0.6 s	121.0 s	4 min
130	8710	2095	166.3 s	35.8 s	150.8 s	0.5 s	3 min	7 min
150	11550	2659	4 min	60.5 s	5 min	0.8 s	7 min	12 min
170	14790	3282	10 min	138.6 s	9 min	0.9 s	12 min	24 min
190	18430	3965	34 min	3 min	12 min	0.9 s	16 min	53 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

Table 4.4.1. Running time of our multivariate attack for HSSP_n.

Comparison with Nguyen-Stern

We compare the two algorithms in Table 4.4.2. 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 $2^{\Omega(n)}$. However, we need more samples, namely $m \simeq n^2/2$ samples instead of $m = 2n$.

	n	90	110	130	150	170	190	220	250
Nguyen-Stern attack [NS99]	m	180	220	260	300	340	–	–	–
	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

Table 4.4.2. 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.

4.4.2 $\mathbf{HSSP}_{\boldsymbol{n}}^{\boldsymbol{\kappa}}$

Similarly to HSSP_n, we receive as input from the orthogonal lattice attack an LLL-reduced basis of the lattice $\tilde{\mathcal{L}}_{\mathbf{x}}$, then we construct the matrix $\mathbf{E}$ of dimension roughly $n^2/2 \times n^2/2$ and we compute the kernel $\mathbf{K} = \ker \mathbf{E}$. However, we heuristically expect the kernel to have dimension $2n$ instead of n. This heuristic was always verified during our experiment. Subsequently, we apply a bit guessing strategy as described in Algorithm 4.5. Finally, to distinguish the vectors $\mathbf{x}_i$'s and recover the basis with fixed Hamming weight we apply Algorithm 4.6 (FH), of Section 4.3.1.

Depending on the value κ/n, we expect different running times due to Heuristic 3. Experimental results are in Table 4.4.3, 4.4.4 and 4.4.5.

					Step 2			
n	κ	m	$\log Q$	Step 1	Kernel mod 3	Guessing	FH	Total
70	35	2590	771	10.3 s	10.3 s	2.2 s	0.8 s	28.4 s
90	45	4230	1148	21.9 s	27.8 s	3.8 s	1.8 s	64.7 s
110	55	6270	1589	50.8 s	57.1 s	6.4 s	3.3 s	140.4 s
130	65	8710	2090	104.5 s	128.6 s	11.3 s	7.6 s	5 min
150	75	11550	2654	4 min	4 min	14.8 s	11.5 s	8 min
170	85	14790	3276	8 min	6 min	21.2 s	23.3 s	18 min
190	95	18430	3957	23 min	10 min	24.4 s	26.9 s	41 min

Table 4.4.3. Running time of our multivariate attack for HSSP_n^κ with $\kappa = n/2$.

					Step 2			
n	κ	m	$\log Q$	Step 1	Kernel mod 3	Guessing	FH	Total
70	17	2590	770	10.1 s	11.1 s	2.8 s	0.5 s	26.5 s
90	22	4230	1149	21.9 s	27.0 s	4.5 s	1.2 s	59.2 s
110	27	6270	1589	47.7 s	66.6 s	10.1 s	2.4 s	136.5 s
130	32	8710	2091	107.7 s	126.5 s	13.2 s	5.2 s	5 min
150	37	11550	2651	4 min	4 min	20.0 s	9.9 s	9 min
170	42	14790	3276	8 min	6 min	26.7 s	12.9 s	16 min
190	47	18430	3957	15 min	10 min	33.9 s	21.3 s	28 min

Table 4.4.4. Running time of our multivariate attack for HSSP_n^κ with $\kappa = n/4$.

					Step 2			
n	κ	m	$\log Q$	Step 1	Kernel mod 3	Guessing	FH	Total
70	56	2590	771	9.5 s	10.8 s	2.7 s	0.5 s	25.6 s
90	72	4230	1149	20.4 s	27.3 s	5.8 s	1.3 s	59.6 s
110	88	6270	1589	44.9 s	58.2 s	9.2 s	2.4 s	124.4 s
130	104	8710	2090	92.8 s	112.3 s	17.4 s	4.7 s	4 min
150	120	11550	2655	4 min	4 min	26.7 s	8.5 s	9 min
170	136	14790	3272	6 min	6 min	28.7 s	16.0 s	15 min
190	152	18430	3957	14 min	9 min	37.5 s	17.1 s	26 min

Table 4.4.5. Running time of our multivariate attack for HSSP_n^κ with $\kappa = 4n/5$.

4.5 Conclusions and remarks

We described our algorithm for solving the hidden subset sum problem in polynomial time, by reducing the binary vectors' search to solving a multivariate polynomial system. We observed that our algorithm heuristically requires $m \approx n^2$. Thus, in the first part of this chapter we described several improvements for the orthogonal lattice attack of Nguyen and Stern. These allowed us to make the algorithm practical, reducing the full asymptotic complexity from $\mathcal{O}(n^{16})$ to $\mathcal{O}(n^9)$. Moreover, as a consequence of those improvements, we could also revise the lower bound on the size of the modulus Q, showing that it only depends on the subset sum density, under reasonable assumptions. In the second part of the chapter, we explained how to derive a quadratic multivariate polynomial system from the output of the orthogonal lattice attack, and we described a method for solving such a system, based on a linearisation technique. We noticed that we can expect two different situations, depending on the distribution of the binary vectors. Consequently, we constructed two algorithms using heuristic assumptions, formulated *ad hoc*. Finally, we also described an algorithm for retrieving fixed Hamming weight bases of random instances of HSSP_n^κ, exploiting statistical properties.

Our contributions presented in this chapter clarified several open questions about the hardness of the hidden subset problem; however, they introduced new issues and open questions, too. Here, we collect some of them, together with a brief discussion and some remarks.

- Both ours and the Nguyen-Stern algorithm consist of two steps. The main difference among them is the second step. Indeed, in this chapter we described an alternative algorithm for the binary vectors' search, that allowed us to solve the problem in polynomial time, but only heuristically. Thus, as a first question, we can consider the problem of finding another strategy for performing the second step but in provable polynomial time. Actually, we have been able to partially address this question. Indeed, in Chapter 5 we will describe an algorithm fulfilling this purpose, but only for random instances of HSSP_n. Moreover, in Section 6.5.1 we will also present a heuristic version of such an algorithm, more efficient both in terms of time and space compared to the multivariate attack.

- As noticed above, our algorithm works under some assumptions and heuristics. Although we found that all of them were always verified in practice, we believe that further investigating them could lead us to tighter rigorous conditions regarding the hardness of HSSP.
- We remark that a multivariate quadratic polynomial system like (4.1) can be derived from any super lattice of $\mathcal{L}_{\mathbf{x}}$. This implies that we do not strictly need the first step to recover the full $\bar{\mathcal{L}}_{\mathbf{x}}$, and so we can consider the orthogonal lattice attack successful if its recovers lattice a generic super lattice of $\mathcal{L}_{\mathbf{x}}$, even if strictly contained in $\bar{\mathcal{L}}_{\mathbf{x}}$. Moreover, notice that we only required $m \approx n^2$ in order to solve (4.1) in polynomial time. In fact, for any value of m, the system (4.1) can be solved, and the binary vectors computed; namely, other methods, *e.g.* Gröbner bases, can be used independently on the number of equations. However, the complexity rapidly grows when we decrease m. In the appendix of the full version of [CG20a], we propose a method for solving (4.1) applicable when a few equations are missing, based on a hybrid strategy between the eigenspaces' computation and the bit-guessing we described in this chapter.
- Finally, our theoretical analysis and the result of the practical experiments in Section 4.4 show that the most time consuming part of the algorithm is the first step. In practice, the improvements we proposed revealed themselves essential to reach large values of n. Therefore, an interesting question is how to further improve the first step. Furthermore, Notarnicola and Wiese in [NW21] recently revised the orthogonal lattice attack of Nguyen and Stern and proposed an alternative method for disclosing a *hidden lattice*. Then, we believe that it would be fruitful to study the impact of their contribution on the understanding of the hardness of HSSP.

5

Our statistical algorithm for HSSP$_n$

We presented in Chapter 4 our polynomial time algorithm for solving HSSP. This algorithm heuristically solves the problem, reducing the binary vectors' search to solving a quadratic multivariate polynomial system; this requires $m \approx n^2$, instead of $m = 2n$ as for the original Nguyen-Stern attack. Essentially, using a sample of size quadratic in n, instead of linear, allows us to drop the complexity from exponential to polynomial.

Since our multivariate algorithm is heuristic only, the purpose of this chapter is to show that there exists an algorithm for provably solving the hidden subset sum problem. As previously discussed, the orthogonal lattice attack works in polynomial time, so we describe here only another algorithm for retrieving the binary vectors. Namely, we illustrate a *statistical learning technique*, reducing the problem of revealing the binary vectors of a random instance of HSSP$_n$ to solving an instance of the *hidden parallelepiped problem* introduced by Nguyen and Regev in [NR09]. The statistical method as presented in this chapter is *proven* to be successful with constant probability, but requires a very large number of samples. Therefore, in Section 6.2 we will show that this technique can be adapted for heuristically solving with fewer samples both HSSP$_n$ and the hidden linear combination problem, too.

The main idea

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

1. From the sample vector $\mathbf{h}$ and the modulus Q, 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 Q, recover the weights α_i.

Our algorithm uses the same first step as in Nguyen-Stern, improved as described in Section 4.1, but we replace BKZ in the second step by a statistical approach. Namely, we propose here an alternative method for solving Problem 4.1 that exploits the properties of random instances of HSSP$_n$.

Recall that we denote by $\mathbf{X} \in \mathbb{Z}^{n \times m}$ the binary matrix of row vectors $\mathbf{x}_i \in \mathbb{Z}^m$, and that according to Definition 2.1 its components are uniformly at random over $\{0,1\}$. From the first step, we obtain a basis of the lattice $\tilde{\mathcal{L}}_{\mathbf{x}}$, namely a matrix of row vectors $\mathbf{C} \in \mathbb{Z}^{n \times m}$ such that $\mathbf{X} = \mathbf{W} \cdot \mathbf{C}$ for some unknown matrix $\mathbf{W} \in \mathbb{Z}^{n \times n} \cap \mathrm{GL}_n(\mathbb{Q})$. Therefore, the matrix $\mathbf{V} = \mathbf{W}^{-1}$ is well defined and belongs to $\mathrm{GL}_n(\mathbb{Q})$. Moreover, we have that $\mathbf{c}_i = \mathbf{V}\hat{\mathbf{x}}_i$ for each pair $(\mathbf{c}_i, \hat{\mathbf{x}}_i)$ of columns of $\mathbf{C}$ and $\mathbf{X}$, respectively.

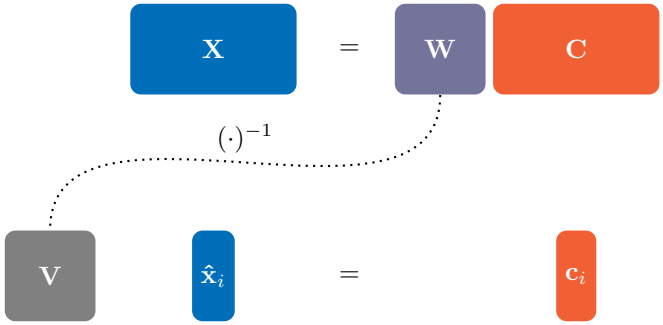


Fig. 5.1. Relation between the columns of $\mathbf{X}$ and $\mathbf{C}$.

For the second step, our main observation is that the m columns of $\mathbf{C}$ belong to the *discrete parallelepiped* associated to $\mathbf{V}$:

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

where $\mathbf{v}_1, \ldots, \mathbf{v}_n \in \mathbb{Q}^n$ are the columns of $\mathbf{V}$. In other words, the discrete parallelepiped associated to $\mathbf{V}$ is the set of all vectors that can be written as $\mathbf{Vx}$ with $\mathbf{x} \in \{0,1\}^n$.

Therefore, our approach consists in recovering the matrix $\mathbf{V}$ from a sample obtained from $\mathbf{C}$, by using a statistical learning 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 this statistical approach allows us to solve HSSP_n 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 associated 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 5.1 (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 5.1 (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} *vectors uniformly sampled from 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 a set of random vectors independently sampled from the finite parallelepiped $\mathcal{P}_{\{0,1\}}(\mathbf{V})$, whereas the Nguyen-Regev algorithm uses elements 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 5.2 (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 vectors sampled 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 5.2) is to reduce to the continuous hidden parallelepiped problem (Problem 5.1). Namely, we show that we can easily generate a sample from the continuous distribution from a sample from the discrete distribution. Then we can use the Nguyen-Regev learning algorithm as a black-box. See Figure 5.2. 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.

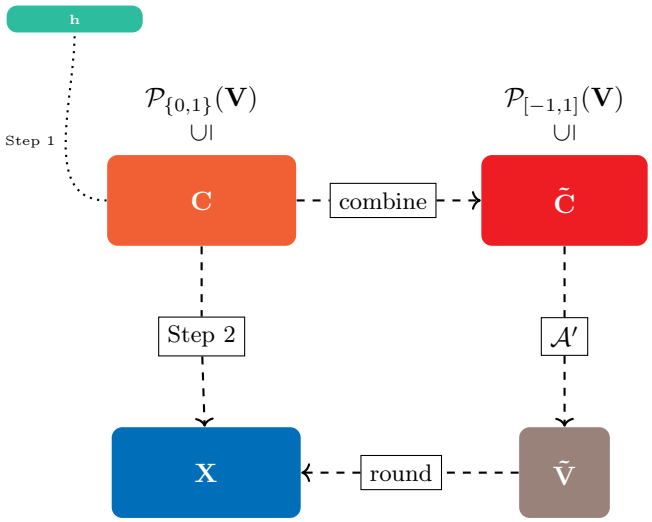


Fig. 5.2. Our algorithm for HSSP_n based on statistical learning.

5.1 From discrete to continuous

Our goal is to obtain a set of statistically independent vectors from the continuous parallelepiped $\mathcal{P}_{[-1,1]}(\mathbf{V})$ from a set of vectors independently sampled uniformly from the discrete parallelepiped $\mathcal{P}_{\{0,1\}}(\mathbf{V})$. Throughout this chapter, the fact that a random variable s is defined as uniformly distributed over a set S is denoted by $s \leftarrow S$; similarly, $\mathbf{s} \leftarrow S$ indicates that $\mathbf{s}$ is a random vector uniformly distributed over S.

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 $y = \sum_{i=1}^{k} b_i 2^{-i}$. To approximate the continuous distribution $x \leftarrow [-1,1]$, we can use one more bit b_0, with $y = -b_0 + \sum_{i=1}^{k} b_i 2^{-i}$. 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 vector $\mathbf{e}$ with continuous components.

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

$$\mathbf{v} = -\mathbf{u}_0 + \sum_{i=1}^{k} 2^{-i}\mathbf{u}_i + \mathbf{e} \tag{5.1}$$

where $\mathbf{u}_i \leftarrow \mathcal{U}(\mathcal{P}_{\{0,1\}}(\mathbf{V}))$ for $0 \le i \le k$ 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 formed as in (5.1). By definition, the vector $\mathbf{v}$ from $\mathcal{D}(\mathbf{V}, k, \varepsilon)$ can be written 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 uniformly random over $\{0, 1\}$, 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 sampled uniformly over 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 a space E as

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

Lemma 5.2. *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 $\mathbf{e} \leftarrow [0, \varepsilon)^n$. 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.

□

5.2 The Nguyen-Regev learning technique

As recalled in Theorem 5.1, the Nguyen-Regev algorithm recovers an approximation of a single column of $\pm\mathbf{V}$, from polynomially many vectors from $\mathcal{U}(\mathcal{P}_{[-1,1]}(\mathbf{V}))$.

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 $\mathbf{L}^{-1}$.

The main intuition behind the algorithm is that when the input is a parallelepiped $\mathcal{P}_{[-1,1]}(\mathbf{A})$ associated to an orthogonal matrix $\mathbf{A}$, 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 depending on $\mathcal{U}(\mathcal{P}_{[-1,1]}(\mathbf{A}))$. Thus, the algorithm first applies an invertible linear transformation in order to get samples from such a parallelepiped, and then performs an iterative algorithm to compute one of the $\mathbf{a}_i$'s. See [NR09].

Repeating the Nguyen-Regev 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 remaining target vectors must belong to $Span\,(V)^{\perp}$. Therefore, at each iterative step we can project on such a space. Although projecting does not work when $\mathbf{V}$ is not orthogonal, this idea can be integrated directly inside the Nguyen-Regev algorithm. Indeed, 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 5.1.

Corollary 5.3. *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 $\pm\mathbf{e}_i$.*

5.3 Our algorithm based on statistical learning

In this section we describe our main algorithm for solving a random HSSP_n based on statistical learning. This comprises the following operations: first we use the block orthogonal lattice attack to compute a matrix $\mathbf{C}$ such that there exists $\mathbf{V} \in \mathrm{GL}_n(\mathbb{Q})$ such that $\mathbf{C} = \mathbf{V}\mathbf{X}$; we extract from $\mathbf{C}$ a sample from the discrete parallelepiped $\mathcal{P}_{\{0,1\}}(\mathbf{V})$ and compose these vectors to produce a sample from a continuous parallelepiped $\mathcal{P}_{[-1,1]}(\mathbf{V})$; then, we apply the Nguyen-Regev statistical learning technique to compute an approximation of $\mathbf{V}$; finally, we recover the binary vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n$ via rounding. See Figure 5.2

We recall that the blocks-variant of the Nguyen-Stern orthogonal lattice attack of Section 4.1.1 computes $\tilde{\mathcal{L}}_{\mathbf{x}} \subset \mathbb{Z}^m$ by running several times the original algorithm with smaller blocks of dimensions $2n$. Specifically, suppose $m = (b+1) \cdot n$ for $b > 0$, we can divide $\mathbf{h}$ and $\mathbf{X}$ in $b+1$ blocks of dimension n, *i.e.* $\mathbf{h} = [\mathbf{h}_0, \ldots, \mathbf{h}_b]$ and $\mathbf{X} = [\mathbf{X}_0, \mathbf{X}_1, \mathbf{X}_2, \cdots, \mathbf{X}_b]$, respectively. Algorithm 4.1 runs Algorithm 3.1 on each one of the b sub-vectors of the form $(\mathbf{h}_0, \mathbf{h}_i) \in \mathbb{Z}^{2n}$ for $1 \leq i \leq b$, and outputs a matrix $\mathbf{C} = [\mathbf{C}_0, \mathbf{C}_1, \mathbf{C}'_2, \cdots, \mathbf{C}'_b]$. In Proposition 4.1 we proved that if we assume that the projection of the lattice $\mathcal{L}_{\mathbf{x}} \subseteq \mathbb{Z}^m$ over the first n components has

rank n, and the projection of $\mathcal{L}_{\mathbf{x}}$ over the first $2n$ coordinates is complete, then we get $\mathbf{C}$ a basis of $\bar{\mathcal{L}}_{\mathbf{x}}$. This implies that $\mathbf{X} = \mathbf{W} \cdot \mathbf{C}$ for some unknown matrix $\mathbf{W} \in \mathbb{Z}^{n \times n} \cap \mathrm{GL}_n(\mathbb{Q})$. As input for the statistical learning, we need samples from a parallelepiped associated to a matrix $\mathbf{V} \in \mathrm{GL}_n(\mathbb{Q})$ such that $\mathbf{C} = \mathbf{V}\mathbf{X}$. In particular, here we do not actually need that $\mathbf{C}$ is a basis of $\bar{\mathcal{L}}_{\mathbf{x}}$ and $\mathbf{W} \in \mathbb{Z}^{n \times n}$, since $\mathbf{C}$ can have rational coefficients, but only that $\mathbf{C} = \mathbf{W}^{-1}\mathbf{X}$. Now, the purpose of the additional hypotheses of Proposition 4.1 on the completion of the projections over the first $2n$ is to guarantee that $\mathbf{C}$ is integral; this implies that here this assumption can actually be avoided. Namely, we have that Algorithm 4.1 returns a valid input for the statistical learning technique with probability at least $1/2$ over the choice of $\boldsymbol{\alpha}$ if Q is a prime of bitsize at least $4n^2(\log n + 1)$.

Thus, let $\mathbf{V} \in \mathrm{GL}_n(\mathbb{Q})$ be the unique matrix such that $[\mathbf{C}_0, \mathbf{C}_1] = \mathbf{V}[\mathbf{X}_0, \mathbf{X}_1]$, this also implies that $\mathbf{C}'_i = \mathbf{V}\mathbf{X}_i$ for $2 \leq i \leq b$. Therefore, we can interpreted the set $\mathfrak{C}$ of all the columns of $\mathbf{C}' = [\mathbf{C}'_2, \cdots, \mathbf{C}'_b]$ as a sample from the uniform distribution over the discrete parallelepiped associated to $\mathbf{V}$, *i.e.* $\mathcal{P}_{\{0,1\}}(\mathbf{V})$. Indeed, the columns of $\mathbf{X}$ are sampled independently and uniformly from $\{0,1\}^n$. Now, given $\mathfrak{C}$ we can generate a set $\mathfrak{V}$ of independent vectors from $\mathcal{P}_{[-1,1]}(\mathbf{V})$ as explained in Section 5.1, and use $\mathfrak{V}$ as input for the Nguyen-Regev algorithm to compute an approximation $\tilde{\mathbf{V}}$ of $\mathbf{V}$. Finally, the $\mathbf{x}_i$'s can be recovered as rounding of $\tilde{\mathbf{V}}^{-1}\mathbf{C}$. We summarise this strategy in Algorithm 5.1, where we denote Algorithm 4.1 by $\mathsf{BlocksOrthoLat}$.

Algorithm 5.1 Statistical attack for HSSP_n

Input: $\mathbf{h}, Q, n$

Output: the vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n$.

1: $[\mathbf{C}_0, \mathbf{C}_1, \mathbf{C}'_2, \cdots, \mathbf{C}'_b] \leftarrow \mathsf{BlocksOrthoLat}(\mathbf{h}, n, m)$
2: Generate a sample $\mathfrak{V}$ of $\mathcal{U}(\mathcal{P}_{[-1,1]}(\mathbf{V}))$ by using the columns of $\mathbf{C}'$ as shown in Section 5.1.
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** $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$

We want to prove that our strategy based on the Nguyen-Regev statistical learning technique solves HSSP_n in polynomial time for certain choices of parameters. Namely, we will show the following fact:

Theorem 5.4. *There exist constant $n_0 \geq 0$ and $\ell > 0$ such that for any $n \geq n_0$, and for any prime integer Q of bitsize at least $4n^2 \log(n+1)$, Algorithm 5.1 solves any random instance of* HSSP_n *with constant probability in polynomial time, using $m = n^\ell$.*

In order to prove Theorem 5.4, we have to show that any lattice $\mathcal{L}_{\mathbf{x}}$ satisfies our hypothesis on the rank with at least constant probability.

Lemma 5.5. *The projection of any lattice $\mathcal{L}_{\mathbf{x}}$ associated to a random* HSSP_n *over the first n components has full rank n with probability at least e^{-2}.*

Proof. 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 projection of $\mathcal{L}_{\mathbf{x}}$ 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, then

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

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 projection of the lattice $\mathcal{L}_{\mathbf{x}}$ over the first n components has full rank n with at least constant probability.

□

Proof (of Theorem 5.4). From Lemma 5.5 we get that the projection of the lattice $\mathcal{L}_{\mathbf{x}} \in \mathbb{Z}^m$ over the first n components is full rank with constant probability. In such a case, as discussed above, we have that the set $\mathfrak{C}$ of all the columns of $\mathbf{C}' = [\mathbf{C}'_2, \cdots, \mathbf{C}'_b]$ is a sample of independent vectors from the uniform distribution over the discrete parallelepiped $\mathcal{P}_{\{0,1\}}(\mathbf{V})$, for $\mathbf{V} \in \mathrm{GL}_n(\mathbb{Q})$ the unique matrix such that $[\mathbf{C}_0, \mathbf{C}_1] = \mathbf{V}[\mathbf{X}_0, \mathbf{X}_1]$. Indeed, the matrix $\mathbf{V}$ is fixed by the first iteration of Algorithm 4.1 and determined by the first $2n$ components of $\mathbf{X}$. We recall that the columns of $\mathbf{X}$ are sampled independently and uniformly from $\{0,1\}^n$ by Definition 2.1; consequently, the columns of $\mathbf{C}'$ are an actual sample from $\mathcal{P}_{\{0,1\}}(\mathbf{V})$, because $[\mathbf{X}_2, \cdots, \mathbf{X}_b]$ is independent from $\mathbf{V}$.

Therefore, the first step of Algorithm 5.1 provides a sample $\mathfrak{C}$ of size $m_0 = m - 2n$, usable to generate a set $\mathfrak{V}$ of $m' = m_0/k$ vectors as explained in Section 5.1. Lemma 5.2 implies that, for k sufficient large, $\mathfrak{V}$ is indistinguishable from a sample of $\mathcal{U}(\mathcal{P}_{[-1,1]}(\mathbf{V}))$. Hence, we can apply Corollary 5.3 with $c_0 = 2$ and a sample of size $m' = n^{c_1}$ from $\mathcal{P}_{[-1,1]}(\mathbf{V})$. In this way, we 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 3 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, *i.e.* the respective columns have ℓ_2 distance smaller than ε. Without loss of generality, we can assume $\mathbf{P} = \mathbf{I}$. Thus, it holds that $\|\tilde{\mathbf{E}} - \mathbf{I}\|_{\max} < \varepsilon$, 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, with $\tilde{\mathbf{W}} = \tilde{\mathbf{V}}^{-1}$, we obtain:

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

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 $\|\mathbf{Q}\|_\infty < 1$, we have $(\mathbf{I} - \mathbf{Q})^{-1} = \sum_{j\geq 0} \mathbf{Q}^j$. 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.

Summarising, we proved that a sample of size $m' = \mathsf{poly}(n)$ of $\mathcal{U}(\mathcal{P}_{[-1,1]}(\mathbf{V}))$ is sufficient for applying Nguyen-Regev attack and computing the binary hidden vectors, and that this can be produced by $m = k \cdot m' + 2n$ elements. Now, notice that k can be chosen 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 vectors m polynomial in n is enough, *i.e.* there exists $\ell > 0$ such that we can solve the problem if $m = n^\ell$, within some constant probability. Moreover, this implies that the time complexity of Algorithm 5.1 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]. Here, 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 all polynomial in n.

□

5.4 Conclusions and remarks

We presented in this chapter our new algorithm for solving HSSP_n. This algorithm consists of two steps: first we disclose a basis of a hidden lattice containing $\mathcal{L}_{\mathbf{x}}$ by using the block orthogonal lattice attack, and then we process such a matrix in order to obtain a sample from the a hidden parallelepiped, so that we can apply the Nguyen-Regev statistical learning attack [NR09] to recover the binary vectors $\mathbf{x}_i$'s. Recall that the original Nguyen-Stern algorithm for the hidden subset sum problem has exponential complexity, as explained in Chapter 3, while the multivariate attack we proposed in Chapter 4 is polynomial-time, but only heuristic. The statistical method presented in this chapter is proven to be successful in polynomial time. This new attack requires a very large value of m. Therefore, in Section 6.5 we will show that this

technique can be adapted for heuristically solving with fewer samples both HSSP_n and an extension of the hidden subset sum problem.

Finally, we remark that the statistical learning attack we proposed is not applicable to HSSP_n^κ. Indeed, the Nguyen-Regev strategy considered in Section 5.2 relies on the assumption that the components of the matrix $\mathbf{X}$ are sampled independently, while for random instances of HSSP_n^κ its columns must have fixed Hamming weight. Therefore, the problem of proving that there exists a polynomial time algorithm for this latter case remains still unsolved. An interesting question is then understanding if it is possible either to adapt this strategy or find another path in order to close this theoretical gap between HSSP_n and HSSP_n^κ.

6

Variants of the hidden subset sum problem

The purpose of this chapter is to extend the study of the hidden subset sum problem to two variants, having application in cryptanalysis.

The hidden subset sum problem naturally arises during the cryptanalysis of the BPV generator [BPV98]; see Section 1.6. Nguyen and Stern in [NS99] notice that, depending on the application and the security notion, the correct problem to consider may be the *affine hidden subset problem* (Definition II) rather than the HSSP. Nevertheless, they showed that their orthogonal lattice attack can be slightly modified in order to produce a basis of $\bar{\mathcal{L}}_{\mathbf{x}}$ in this case, too. This implies that our algorithms can still be applied to solve this variant, since as input for our binary vectors' search we take $\bar{\mathcal{L}}_{\mathbf{x}}$, too. Thus, in Section 6.1 we recall the affine problem and we provide our analysis of the *affine orthogonal lattice attack*.

The second variant we analyse is the *hidden linear combination problem* (Definition III), which is related to the security of the *extended BPV generator* (EBPV) by Nguyen, Stern and Shparlinski, described in [NSS01]. Essentially, Nguyen *et al.* proposed to include in the BPV generator small exponents, *i.e.* given an hidden set of precomputed pairs $\{(\alpha_i, g^{\alpha_i} \pmod p)\}$, they suggest to produce new pairs by performing linear combinations of the exponents: a pair (x, g^x) is compute as $x = \sum_{i=1}^{n} \alpha_i \cdot x_i \bmod q$ and so

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

where $x_i \in \{0, \ldots, B\}$. See Section 1.6 for details. Hence, the second part of this chapter is devoted to the study of the hidden linear combination problem. Specifically, in Section 6.2 we first introduce the problem and summarise our contributions; then, in Section 6.3 we discuss the notion of *random* instance and its relation with HSSP. Regarding the attacks, first in Section 6.4 we analyse the extension of the Nguyen-Stern lattice based algorithm, then in Section 6.5 we describe a heuristic variant of the *statistical learning attack* for solving the hidden linear combination problem in polynomial time. Finally, we provide the result of our experiments, which shows that this algorithm is also quite efficient in practice.

6.1 The affine hidden subset sum problem

In this section we analyse the affine hidden subset problem, which is a variant of the hidden subset sum problem introduced in [NS99]. Namely, let Q be an integer and let $s, \alpha_1, \ldots, \alpha_n$ be random integers in $\mathbb{Z}_Q$; consider $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{Z}^m$ with components in $\{0, 1\}$ and $\mathbf{e} = (e_1, \ldots, e_m)$ a vector with random components in $[0, 2^t[$ for $t \in \mathbb{N}$. Let $\mathbf{h} = (h_1, \ldots, h_m) \in \mathbb{Z}^m$ satisfying:

$$\mathbf{h} + s\mathbf{e} = \alpha_1\mathbf{x}_1 + \alpha_2\mathbf{x}_2 + \cdots + \alpha_n\mathbf{x}_n \pmod{Q}. \tag{6.1}$$

Given Q, $\mathbf{h}$ and $\mathbf{e}$, the problem consists in recovering both s, 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.

In [NS99], Nguyen and Stern observed that for the cryptanalysis of the embedding of the BPV generator in some applications, the correct problem to consider is the affine hidden subset sum problem. Indeed, they describe as an example a passive attack against the generator used in Schnorr's signatures. We recall this attack in Section 1.6. The authors also explain how to modify their algorithm to address this variant, too. Namely, they show that the orthogonal lattice attack can be adapted in order to first compute a lattice containing both the $\mathbf{x}_i$'s and $\mathbf{e}$ and then extract a basis of $\bar{\mathcal{L}}_{\mathbf{x}}$; subsequently, they suggest to apply the second step as in the original problem. The second step of this strategy, therefore, has the same issues discussed for the original attack. Nevertheless, we can directly apply the same countermeasures described in Chapter 4, since the second step is independent from the error vector.

Thus, in Section 6.1.1 we describe the affine orthogonal lattice attack by Nguyen and Stern and then we discuss its correctness. This analysis is included in the full version of [CG20a].

6.1.1 Affine orthogonal lattice attack

We denote by $\mathcal{L}_{\mathbf{x},\mathbf{e}}$ the lattice generated by $\mathbf{e}$ and the vectors $\mathbf{x}_i$'s, and by $\mathcal{L}_{\mathbf{x}}$ the lattice generated by the vectors $\mathbf{x}_i$'s only. The goal of this section is to show that the first step of the Nguyen-Stern algorithm can be modified for computing a basis of $\bar{\mathcal{L}}_{\mathbf{x}}$. The Nguyen-Stern *affine orthogonal lattice attack* is summarised by Algorithm 6.1. We present the attack here in a slightly different (but equivalent) form, compared to that in [NS99], which is more suitable for our analysis.

Algorithm 6.1 Orthogonal lattice attack (affine variant) [NS99]

Input: $\mathbf{h}, \mathbf{e}, Q, n, m$.

Output: A basis of $\bar{\mathcal{L}}_{\mathbf{x}}$.

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

As for the original orthogonal lattice attack described in Chapter 3, we can prove that Algorithm 6.1 recovers a basis of $\bar{\mathcal{L}}_{\mathbf{x}}$ in polynomial time, under certain conditions on the lattice $\mathcal{L}_{\mathbf{x},\mathbf{e}}$. Recall that once $\bar{\mathcal{L}}_{\mathbf{x}}$ has been recovered, one can apply either the second step of the Nguyen-Stern algorithm to recover the $\mathbf{x}_i$'s in exponential time, or one of our proposals, depending on the distribution of the binary vectors. Then from the $\mathbf{x}_i$'s one can eventually recover s and the α_i's by solving a linear system.

Retracing our analysis of Section 3.2 of the orthogonal lattice attack, we first prove a rigorous condition on the lattice minima.

Lemma 6.1. *Assume that the lattice $\mathcal{L}_{\mathbf{x},\mathbf{e}}$ has rank $n+1$. Algorithm 6.1 computes at Step 3 a basis of the lattice $\bar{\mathcal{L}}_{\mathbf{x},\mathbf{e}}$ in polynomial time under the condition $m > n$ and*

$$\sqrt{mn} \cdot 2^{\frac{m}{2}} \cdot \lambda_{m-n-1}\left(\mathcal{L}_{\mathbf{x},\mathbf{e}}^{\perp}\right) < \min\left(\lambda_1\left(\Lambda_Q^{\perp}(\boldsymbol{\alpha})\right), Q2^{-t}\right). \tag{6.2}$$

Proof. As previously, the main observation is that if $\langle \mathbf{u}, \mathbf{h} \rangle \equiv 0 \pmod{Q}$ and $\langle \mathbf{u}, \mathbf{e} \rangle \equiv 0 \pmod{Q}$, then we obtain:

$$\langle \mathbf{u}, \mathbf{h} \rangle + s\langle \mathbf{u}, \mathbf{e} \rangle \equiv \alpha_1 \langle \mathbf{u}, \mathbf{x}_1 \rangle + \cdots + \alpha_n \langle \mathbf{u}, \mathbf{x}_n \rangle \equiv 0 \pmod{Q} \tag{6.3}$$

and therefore 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 $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$ modulo Q. As previously, when the vector

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

is shorter than the shortest vector orthogonal to $\boldsymbol{\alpha}$ modulo Q we must have $\mathbf{p}_{\mathbf{u}} = 0$, and therefore $\mathbf{u} \in \mathcal{L}_{\mathbf{x}}^{\perp}$. Since $\|\mathbf{p}_{\mathbf{u}}\| \leq \sqrt{mn}\|\mathbf{u}\|$, given any $\mathbf{u} \in \Lambda_Q^{\perp}(\mathbf{h}, \mathbf{e})$ we must have $\mathbf{u} \in \mathcal{L}_{\mathbf{x}}^{\perp}$ under the condition:

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

Moreover, given $\mathbf{u} \in \mathcal{L}_0'$, we have $\langle \mathbf{u}, \mathbf{e} \rangle = 0 \pmod{Q}$. If $\mathbf{u}$ and $\mathbf{e}$ are short enough, the equality will hold over $\mathbb{Z}$. More precisely, if $\|\mathbf{u}\| \cdot \|\mathbf{e}\| < Q$, we must have $\langle \mathbf{u}, \mathbf{e} \rangle = 0$. This means that if $\|\mathbf{u}\|\sqrt{m}2^t < Q$ we must have $\langle \mathbf{u}, \mathbf{e} \rangle = 0$.

The lattice $\mathcal{L}_0' = \Lambda_Q^{\perp}(\mathbf{h}, \mathbf{e})$ is full rank of dimension m, since it contains $Q\mathbb{Z}^m$. Moreover given a vector $\mathbf{u} \in \mathcal{L}_{\mathbf{x},\mathbf{e}}^{\perp}$, from (6.3) we must have $\langle \mathbf{u}, \mathbf{h} \rangle = 0 \pmod{Q}$; therefore $\mathcal{L}_{\mathbf{x},\mathbf{e}}^{\perp} \subseteq \mathcal{L}_0'$. Now, consider $\mathbf{u}_1, \ldots, \mathbf{u}_m$ an LLL-reduced basis of $\mathcal{L}_0'$. For each $j \leq m-n-1$ we have

$$\|\mathbf{u}_j\| \leq 2^{\frac{m}{2}} \cdot \lambda_{m-n-1}(\mathcal{L}_0') \leq 2^{\frac{m}{2}} \cdot \lambda_{m-n-1}\left(\mathcal{L}_{\mathbf{x},\mathbf{e}}^{\perp}\right) \tag{6.4}$$

since $\mathcal{L}_{\mathbf{x},\mathbf{e}}^{\perp}$ is a sublattice of $\mathcal{L}_0'$ of dimension $m-n-1$. This implies that when

$$\sqrt{mn} \cdot 2^{\frac{m}{2}} \cdot \lambda_{m-n-1}\left(\mathcal{L}_{\mathbf{x},\mathbf{e}}^{\perp}\right) < \lambda_1\left(\Lambda_Q^{\perp}(\boldsymbol{\alpha})\right)$$

the vectors $\mathbf{u}_1, \ldots, \mathbf{u}_{m-n-1}$ must belong to $\mathcal{L}_{\mathbf{x}}^{\perp}$. Additionally, when:

$$\sqrt{m} \cdot 2^{m/2+t} \cdot \lambda_{m-n-1}\left(\mathcal{L}^{\perp}_{\mathbf{x},\mathbf{e}}\right) < Q$$

the vectors $\mathbf{u}_1, \ldots, \mathbf{u}_{m-n-1}$ are also orthogonal to $\mathbf{e}$, so they must belong to $\mathcal{L}^{\perp}_{\mathbf{x},\mathbf{e}}$.

This means that $\langle \mathbf{u}_1, \ldots, \mathbf{u}_{m-n-1} \rangle$ is a full rank sublattice of $\mathcal{L}^{\perp}_{\mathbf{x},\mathbf{e}}$, and therefore we obtain $\langle \mathbf{u}_1, \ldots, \mathbf{u}_{m-n-1} \rangle^{\perp} = \bar{\mathcal{L}}_{\mathbf{x},\mathbf{e}}$. Finally, Algorithm 6.1 is polynomial-time, because both the LLL reduction step of $\mathcal{L}'_0$ and the LLL-based orthogonal computation of $\mathcal{L}^{\perp}_{\mathbf{x},\mathbf{e}}$ are polynomial-time. □

Finally, we show a sufficient condition under which a basis of $\bar{\mathcal{L}}_{\mathbf{x}}$ can be extracted from $\bar{\mathcal{L}}_{\mathbf{x},\mathbf{e}}$.

Lemma 6.2. *Assume that $\mathcal{L}_{\mathbf{x},\mathbf{e}}$ is of rank $n+1$. Let $(\mathbf{c}_1, \ldots, \mathbf{c}_{n+1})$ be an LLL-reduced basis of $\bar{\mathcal{L}}_{\mathbf{x},\mathbf{e}}$. Then $(\mathbf{c}_1, \ldots, \mathbf{c}_n)$ is an LLL-reduced basis of $\bar{\mathcal{L}}_{\mathbf{x}}$, under the condition:*

$$2^{n/2} \cdot \sqrt{m} \cdot m^{n/2} < \det \bar{\mathcal{L}}_{\mathbf{x},\mathbf{e}}.$$

Proof. Since the n vectors $\mathbf{x}_i$ are linearly independent and $\|\mathbf{x}_i\| \leq \sqrt{m}$, we have $\lambda_n(\bar{\mathcal{L}}_{\mathbf{x},\mathbf{e}}) \leq \lambda_n(\mathcal{L}_{\mathbf{x},\mathbf{e}}) \leq \sqrt{m}$. Therefore for all $1 \leq i \leq n$ we have:

$$\|\mathbf{c}_i\| \leq 2^{n/2} \lambda_n(\bar{\mathcal{L}}_{\mathbf{x},\mathbf{e}}) \leq 2^{n/2}\sqrt{m}.$$

We claim that for all $1 \leq i \leq n$, we must have $\mathbf{c}_i \in \bar{\mathcal{L}}_{\mathbf{x}}$. Otherwise, the vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n, \mathbf{c}_i$ generate a full-rank sublattice $\mathcal{L}'$ of $\bar{\mathcal{L}}_{\mathbf{x},\mathbf{e}}$; then from Hadamard's inequality:

$$\det \bar{\mathcal{L}}_{\mathbf{x},\mathbf{e}} \leq \det \mathcal{L}' \leq \|\mathbf{c}_i\| \cdot \prod_{j=1}^{n} \|\mathbf{x}_j\| \leq 2^{n/2}\sqrt{m} \cdot m^{n/2}$$

which contradicts the hypothesis of the lemma. This implies that $(\mathbf{c}_1, \ldots, \mathbf{c}_n)$ is an LLL-reduced basis of $\bar{\mathcal{L}}_{\mathbf{x}}$.

□

Both previous lemmas admit heuristic improvements and evaluations, depending on the generation of the vectors $\mathbf{x}_i$'s and $\mathbf{e}$. We do not specify them here, since a large variety of combinations is possible according to the particular application considered.

6.2 The hidden linear combination problem

In this section we introduce a natural extension of the hidden subset sum problem where the coefficients of the vectors $\mathbf{x}_i$'s lie in a discrete interval $\{0, \ldots, B\}$ instead of $\{0, 1\}$, for a given $B \in \mathbb{N}^+$. Specifically, let Q be a positive integer, and let $\alpha_1, \ldots, \alpha_n$ be integers in $\mathbb{Z}_Q$. Let $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{Z}^m$ be vectors with components in $\{0, \ldots, B\}$ and $\mathbf{h} = (h_1, \ldots, h_m) \in \mathbb{Z}^m$ satisfying:

$$\mathbf{h} = \alpha_1\mathbf{x}_1 + \alpha_2\mathbf{x}_2 + \cdots + \alpha_n\mathbf{x}_n \pmod{Q} \tag{6.5}$$

Given Q, B and $\mathbf{h}$, the *hidden linear combination problem* (HLCP) consists in recovering 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 (Definition III).

We observed that the hardness of this problem relates to the cryptanalysis of the *extended BPV generator* (EBPV), proposed by Nguyen *et al.* in [NSS01]. See Section 1.6. Indeed, including the EBPV in cryptographic protocols generates vulnerabilities. Namely, the attacks proposed for the BPV generator can be similarly applied in this case, too; but in this case such attacks produce instances of the hidden linear combination problem, rather than of HSSP.

The addition of the parameter B modifies the hardness of the problem and causes the impracticability of certain attacks for HLCP. For instance, extending the multivariate attack becomes infeasible also for quite small $B > 1$. While, we can heuristically extend the statistical learning technique of Chapter 5: in Section 6.5 we will present a statistical algorithm for HLCP that is quite efficient in practice. This variant requires fewer samples, and in some cases it works when the Hamming weight is fixed, too. Moreover, for HSSP_n it is faster than the multivariate attack.

6.2.1 Road map and summary of our contributions regarding HLCP

The remaining of this chapter is devoted to the hidden linear combination problem. We introduced this problem in [CG22], not being aware of any similar definition in the literature. The content here presented is an improvement of the results in [CG22].

As done for HSSP, we first formalise, in Section 6.3, the distributions of HLCP problems, coherently with the notions introduced in Section 2.1, and we discuss certain statistical properties of the algebraic entities involved.

Then, we discuss the extension of the known approaches for solving HSSP to HLCP. Specifically, we first describe an extension of the Nguyen-Stern algorithm for solving the HLCP 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 $\mathbf{x}_i$'s. Notice that instead the multivariate approach from [CG20a] does not apply to the extended problem, as it would lead to multivariate polynomials of degree $B + 1$; the technique would then remain polynomial-time only for constant B, and be essentially impractical. Finally, we describe a statistical learning algorithm very close to the Nguyen-Regev algorithm [NR09], that allows us to solve HLCP with a heuristic complexity $\mathsf{poly}(n, B)$. A more detailed summary of Section 6.4 and Section 6.5, respectively, follows.

- *Extending the Nguyen-Stern algorithm.* We show that the Nguyen-Stern algorithm can be adapted to the hidden linear combination problem; however, 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 sample vector $\mathbf{h}$ and the modulus Q, 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 Q, recover the weights α_i.

For the first step, the orthogonal lattice attack can be directly applied with $\mathbf{h}, Q, n$ as input. The main difference here is that the minimal size of the modulus Q, for which we can expect the attack to succeed, depends on B, too. Indeed, we prove that the orthogonal lattice attack recovers a basis of $\bar{\mathcal{L}}_{\mathbf{x}}$ in polynomial time with non-negligible probability when the size of the modulus Q is a prime integer of bitsize at least $2mn \log m + (n+1)n \log B$. This corresponds to an *extended subset sum density* $d = n \log(B+1)/\log Q = \mathcal{O}(1/(n \log n))$.
The second step of Nguyen-Stern attack for the hidden linear combination problem is also essentially the same as for the hidden subset sum problem. As for HSSP, the first step 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, in this case the recovered basis vectors $(\mathbf{c}_1, \ldots, \mathbf{c}_n)$ can be much larger than the original vectors $\mathbf{x}_i$'s. 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 show that we obtain a similar condition on the BKZ approximation factor as in the binary case, and eventually the heuristic running time is $2^{\Omega(n)} \cdot \log^{\mathcal{O}(1)} B$.

- *Our statistical learning strategy.* In Chapter 5 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 sampled from the discrete parallelepiped $\mathcal{P}_{\{0,1\}}(\mathbf{V})$. Similarly, in the hidden linear combination problem, the vectors $\mathbf{x}_i$'s have coordinates uniformly distributed over $\{0, \ldots, B\}$. Then, the m columns of a given basis of $\bar{\mathcal{L}}_{\mathbf{x}}$ can be seen as sampled from the discrete parallelepiped $\mathcal{P}_{\{0,\ldots,B\}}(\mathbf{V}) = \{\mathbf{V} \cdot \mathbf{x} \colon \mathbf{x} \in \{0, \ldots, B\}^n\}$. Therefore, we can reduce the hidden linear combination problem to an analogous of Problem 5.2, where given $\mathsf{poly}(n, B)$ independent samples from the uniform distribution over $\mathcal{P}_{\{0,\ldots,B\}}(\mathbf{V})$, we want to recover the columns of $\mathbf{V}$.
For HSSP_n we described our algorithm where we produce a continuous distribution from a given sample, and then we apply the Nguyen-Regev statistical technique [NR09]. See Chapter 5. However, for HLCP, the former operation is less straightforward and not convenient from a practical point of view. Indeed, it requires the number of samples m to be very large. Here, we show that we can avoid that step, since another statistical learning technique can be directly applied. Namely, instead of producing a continuous distribution and applying Nguyen-Regev result to recover an approximation of the matrix $\mathbf{V}$, it is more efficient in practice to apply a strategy more similar to the FastICA algorithm [HO97]. In fact, the Nguyen-Regev algorithm can be actually seen as an instantiation of the FastICA algorithm, *i.e.* an algorithm for solving the *signal source separation problem*, with kurtosis chosen as the cost function.
Proving that $\mathsf{poly}(n, B)$ samples are sufficient for solving the problem is less straightforward due to the discrete nature of the problem. Therefore, in Section 6.5 we only provide a simple heuristic argument for the complexity of this new variant of the statistical attack.
Finally, we collect some experiments showing that this statistical learning technique based on the FastICA model 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 algorithm; it is also much faster. Indeed, we remark that this new heuristic statistical method does not require us to transform the samples as in Step 2 of Algorithm 5.1, because applying this generic statistical learning algorithm allows us to manage discrete distributions, directly.

6.3 "Random" HLCP

To discuss the hardness of the linear combination problem on *average*, it is crucial to determine how the vectors $\mathbf{x}_i$'s (Definition III) are produced. Thus, the purpose of this section is to specify the notion of *random instance* of HLCP, as we have already done for HSSP in Section 2.1. Following our model for HSSP, we first identify the main parameters of the problem:

n is a positive integer, this is the number of weights $\alpha_1, \ldots, \alpha_n$;
m is a positive integer, this is the dimension of the vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n, \mathbf{h}$;
Q is the modulus of the problem;
B is a positive integer, this is the bound on the coefficients of $\mathbf{x}_i$'s.

We then extend Definition 2.1 and 2.2, respectively. Recall that $\mathcal{B}_n(B) = \{0, \ldots, B\}^n$ and $\mathcal{B}_{n,\kappa}(B)$ is the subset of $\{0, \ldots, B\}^n$ of vectors having Hamming weight exactly κ.

Definition 6.3 ($\mathrm{HLCP}_{n,B}(m, Q)$). *A* random instance *of* $\mathrm{HLCP}_{n,B}(m, Q)$ *is produced by sampling the weights* $\alpha_1, \ldots, \alpha_n$ *uniformly randomly over* $\mathbb{Z}_Q$ *and the vectors* $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{Z}^m$ *with independent components uniform over* $\{0, \ldots, B\}$*, and computing the hidden linear combinations* $\mathbf{h}$ *according to* (6.5).

Definition 6.4 ($\mathrm{HLCP}^{\kappa}_{n,B}(m, Q)$). *A* random instance *of* $\mathrm{HLCP}^{\kappa}_{n,B}(m, Q)$ *is produced as follows:*

1. *the weights* $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$ *are sampled uniformly at random over* $\mathbb{Z}_Q$*;*
2. $\mathbf{X} \in \mathbb{Z}^{n \times m}$ *is a matrix whose columns are vectors sampled independently and uniformly at random from* $\mathcal{B}_{n,\kappa}(B)$*;*
3. $\mathbf{h} = \boldsymbol{\alpha} \cdot \mathbf{X} \pmod{Q}$.
4. *the vectors* $\mathbf{x}_1, \ldots, \mathbf{x}_n$ *are the rows of* $\mathbf{X}$.

Furthermore, we define the *extended density* as $d = n \log(B+1)/\log Q$, and the *sparsity* ς of a hidden linear combination problem as the (matrix) density of the matrix $\mathbf{X}$ of row vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n$, which is the ratio between the number of non zero coordinates of $\mathbf{X}$ and nm. The expectation of the sparsity of a random instance of $\mathrm{HLCP}_{n,B}(m, Q)$ is $B/(B+1)$, while a random instance of $\mathrm{HLCP}^{\kappa}_{n,B}(m, Q)$ has always fixed sparsity $\varsigma = \kappa/n$. An hidden linear combination is said to be *balanced* if $\varsigma \approx B/(B+1)$, and *unbalanced*, otherwise; specifically, it is *sparse* if $\varsigma \ll B/(B+1)$. Notice that these definitions coincide with those given in Section

2.1 when $B = 1$. Indeed, notice $\mathrm{HLCP}^{\kappa}_{n,1}(m, Q) = \mathrm{HSSP}^{\kappa}_{n}(m, Q)$ and $\mathrm{HLCP}_{n,1}(m, Q) = \mathrm{HSSP}_{n}(m, Q)$.

We conclude this section discussing more explicitly these properties and a reduction between HLPC and HSSP.

6.3.1 About density and hardness of HLCP

Let us consider a random instance of $\mathrm{HLCP}^{\kappa}_{n,B}(m, Q)$. Each coordinate h of $\mathbf{h}$ satisfies

$$h = \langle \boldsymbol{\alpha}, \boldsymbol{\epsilon} \rangle \pmod{Q}$$

for some $\boldsymbol{\epsilon} \in \mathcal{B}_{n,\kappa}(B)$. Namely, h is a modular sum, according to the definition given in Section 1.5. Similarly, any coordinate of a sample vector $\mathbf{h}$ of an instance of $\mathrm{HLCP}_{n,B}(m, Q)$ is a modular sum, but with $\boldsymbol{\epsilon} \in \mathcal{B}_{n}(B)$.

Nguyen, Shparlinski and Stern studied the distribution of these modular sums in [NSS01], and they proved in certain cases the distribution of modular sums is statistically close to the uniform distribution over $\mathbb{Z}_Q$. See Section 1.5 for more details. Their results help us to understand for which sets of parameters HLCP can be considered hard. Indeed, similarly to the binary case from Equation (1.6) we obtain that for any $c > 0$ the probability that

$$\Delta_{\alpha}(\mathcal{B}) \leq 2^{c-1} \sqrt{\frac{Q}{|\mathcal{B}|}} \tag{6.6}$$

is at least $1 - 2^{-c}$, where we recall $\Delta_{\alpha}(\mathcal{B})$ denotes the statistical distance between uniform distribution over $\mathbb{Z}_Q$ and the distribution of modular sums for a fixed $\boldsymbol{\alpha}, Q$ and $\mathcal{B}$. For HLCP_n and $\mathrm{HLCP}^{\kappa}_{n}$ we have $|\mathcal{B}_n(B)| = (B+1)^n$ and $|\mathcal{B}_{n,\kappa}(B)| = B^{\kappa} \cdot \lambda$ where $\lambda = \binom{n}{\kappa}$, respectively.

Combining this bound with Lemma 1.17, we have that there exists $c > 0$ such that the statistical distance between the distribution of any $\mathbf{h}$ generated as $\mathrm{HLCP}_n(m, Q)$ (resp. $\mathrm{HLCP}^{\kappa}_{n}(m, Q)$) and a random vector uniformly sampled over $\mathbb{Z}_Q^m$ is lower than

$$m 2^{c-1} \sqrt{\frac{Q}{(B+1)^n}} \quad \left(\text{resp. } m 2^{c-1} \sqrt{\frac{Q}{(B+1)^{\kappa} \lambda}} \right)$$

with probability at least $1 - 2^{-c}$.

This suggests that for polynomial values of m, random instances of $\mathrm{HLCP}_{n,B}(m, Q)$ should be hard when $\log Q \lesssim n \cdot \log(B+1)$. The same holds for balanced random instances of $\mathrm{HLCP}^{\kappa}_{n,B}(m, Q)$. Notice that this is coherent with our observations regarding the relation between density and hardness of the hidden subset sum problem; see Section 2.1.

In Section 6.4, we will show that we can obtain an analogue of Theorem 3.1 for HLCP; namely, we show that if the density of HLCP is sufficiently low, an orthogonal lattice attack can be used to reveal a lattice containing the $\mathbf{x}_i$'s. This suggests that low-density hidden linear

combination problems may be weaker. As a matter of fact, we are able to solve low-density instances in practice with our heuristic statistical learning attack; see Section 6.5.2.

6.3.2 Sparse HLCP and additional vectors

We observed in Section 3.2.2 that any lattice $\bar{\mathcal{L}}_{\mathbf{x}}$ associated to a random HSSP_n^κ contains additional binary vectors, due to the condition of the Hamming weight of the matrix $\mathbf{X}$'s columns. Among the several consequences, we argued that this fact has an impact on the method that has to be chosen for the bounded vectors' search. We explain now that for $B > 1$ the presence of additional vectors is likely to be determined by the sparsity rather than the Hamming weight.

If $B > 1$, it is no longer true that $\mathbf{1}_n \cdot \mathbf{X} = \kappa \cdot \mathbf{1}_m$. Consequently, we do not know *a priori* if $\mathbf{1}_m \in \bar{\mathcal{L}}_{\mathbf{x}}$, and we can only affirm that

$$\|\mathbf{1}_n \cdot \mathbf{X}\|_\infty \leq \kappa B.$$

So, we can not claim the certainty of the presence of additional bounded vectors in $\bar{\mathcal{L}}_{\mathbf{x}}$ by the fixed Hamming weight argument. However, we now show that the sum of any two $\mathbf{x}_i$'s has likely coordinates smaller that B, when the sparsity ς is low; this fact implies that we can expect to have several additional vectors of $\mathcal{L}_{\mathbf{x}}$ lying in $\{0, \ldots, B\}^m$ if $\varsigma = \kappa/n \ll B/(B+1)$.

Given any two coordinate $\tilde{x}_i, \tilde{x}_j$ of $\tilde{\mathbf{x}}$ a generic column of $\mathbf{X}$ as in HSSP_n^κ, we have that

$$\mathbb{E}[\tilde{x}_i] = \frac{B+1}{2}\frac{\kappa}{n} \text{ and } \mathbb{E}[\tilde{x}_i^2] = \frac{(B+1)(2B+1)}{6}\frac{\kappa}{n},$$

Since $\tilde{x}_i\tilde{x}_j \neq 0$ if and only if both variables are nonzero, then

$$\mathbb{E}[\tilde{x}_i\tilde{x}_j] = \frac{(B+1)^2}{4}\frac{\kappa}{n}\frac{\kappa-1}{n-1}.$$

Let us consider $\mathbf{z} = \mathbf{x}_i + \mathbf{x}_j$ for some $i, j \in 1, \ldots, n$, the expectation of its coordinates is $\mathbb{E}[z_j] = 2\frac{\kappa}{n}\frac{B+1}{2}$ and

$$\mathbb{E}[z_j^2] = 2(\mathbb{E}[\tilde{x}_i^2] + \mathbb{E}[\tilde{x}_i\tilde{x}_j]) = \frac{(B+1)}{2}\frac{\kappa}{n}\left(\frac{2B+1}{3} + \frac{B+1}{2}\frac{\kappa-1}{n-1}\right)$$

By using the Chebishev's inequality and union bound, we get that for any $a > 0$

$$\mathbb{P}(|\mathbf{z} - \mathbb{E}[\mathbf{z}]| \geq a) \leq m\frac{\mathbb{E}[z_j^2] - \mathbb{E}[z_j]^2}{a^2}.$$

This implies that for fixed B, the probability of the coordinates of the sum of any two vectors $\mathbf{x}_i$'s being smaller than B is very high if sparsity ς is sufficiently low. Consequently, in such a case identifying a correct basis may be more difficult by using our algorithm; on the other

hand, we have that an exhaustive search of the solution is more effective for sparse instances. In the following we will mainly focus on non-sparse instances.

6.3.3 Reducing HLCP to HSSP

Let us consider any instance of the hidden linear combination problem, either of $\mathrm{HLCP}^{\kappa}_{n,B}(m,Q)$ or $\mathrm{HLCP}_{n,B}(m,Q)$. Given $\mathbf{h}, Q$, the goal is to recover the vector $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n) \in \mathbb{Z}_Q^n$ and the vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \{0, \ldots, B\}^m$ satisfying

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

The purpose of this section is to show that there exists a trivial transformation of each instance of HLCP with n weights into an instance of HSSP with $n(\lfloor \log B \rfloor + 1)$ weights. However, at the current state of the art, this reduction does not lead us to a solution.

Each coefficients $x_{i,j}$ of $\mathbf{x}_i$ has an unique binary decomposition as $x_{i,j} = \sum_{i=0}^{L-1} x_{i,j,k} 2^k$ where $L = \lfloor \log B \rfloor + 1$ and $x_{i,j,k} \in \{0,1\}$. This implies that each $\mathbf{x}_i$ can be written uniquely as sum of binary vectors

$$\mathbf{x}_i = \mathbf{x}_{i,0} + 2\mathbf{x}_{i,1} + \cdots + 2^L \mathbf{x}_{i,n}$$

Therefore, there exist $s = nL$ binary vectors $\mathbf{t}_1, \ldots, \mathbf{t}_s \in \{0,1\}^m$ such that

$$\mathbf{h} = \beta_1 \mathbf{t}_1 + \beta_2 \mathbf{t}_2 + \cdots + \beta_s \mathbf{t}_s \pmod{Q}$$

Thus, given an instance of the hidden linear combination problem, we obtained an instance of the hidden subset sum problem.

Superficially, we might apply the results we proved in the previous chapters to determine a bound on the parameters. In particular, we might erroneously conclude that if the lattice $\mathcal{L}_{\mathbf{t}}$ has rank s, we can recover a basis of $\bar{\mathcal{L}}_{\mathbf{t}}$ in polynomial time for values of Q of size greater than $2sn \log s$. However, this would be a misuse of Theorem 3.1. Indeed, this theorem is based on the fact that the orthogonal lattice attack recovers a basis of $\overline{\mathcal{L}}_{\mathbf{t}}$ under the condition

$$\sqrt{ms} \cdot 2^{\frac{m}{2}} \cdot \lambda_{m-s}\left(\mathcal{L}_{\mathbf{t}}^{\perp}\right) < \lambda_1\left(\Lambda_Q^{\perp}(\boldsymbol{\beta})\right), \tag{6.7}$$

but in this case lattice $\Lambda_Q^{\perp}(\boldsymbol{\beta})$ contains vectors of the form $(0 \ldots, 0, 2, 0, \ldots, 0, -1, 0, \ldots, 0)$, since $\boldsymbol{\beta}$ is a permutation of the vector $(\boldsymbol{\alpha}, 2\boldsymbol{\alpha}, \ldots, 2^L\boldsymbol{\alpha})$. This implies that in the lattice we have vectors whose norm is $\sqrt{5}$, and the condition (3.2) is never verified. Namely, there is no guarantee of succeeding. Therefore, at the current state of the art, the main bottleneck for using such a reduction algorithm resides in the first step of the hidden subset sum solver, because the orthogonal lattice approach does not solve the underlying hidden lattice problem. The practical experiments we performed always failed, confirming the ineffectiveness of this approach. Nevertheless, this further clarifies the relationship between the hidden subset sum problem and the hidden linear problem.

6.4 Extending the lattice attack for HLCP

The first attack for solving the hidden subset sum algorithm is due to Nguyen and Stern, who described a lattice based algorithm in [NS99]. We illustrated and analysed this algorithm in Section 3.1. Here, we analyse a straightforward extension of that algorithm for solving the hidden linear combination problem. The present section follows and enhances the analysis of [CG22].

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

$$\mathbf{h} = \alpha_1 \mathbf{x}_1 + \alpha_2 \mathbf{x}_2 + \cdots + \alpha_n \mathbf{x}_n \pmod{Q} \tag{6.8}$$

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 sample vector $\mathbf{h}$ and the modulus Q, 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 Q, recover the weights α_i.

These are implemented as an orthogonal lattice attack and an application of BKZ, respectively.

6.4.1 First step: orthogonal lattice attack

The goal of the first step of the Nguyen-Stern attack is to recover the hidden lattice $\bar{\mathcal{L}}_{\mathbf{x}} \subseteq \mathbb{Z}^m$, *i.e.* the completion of the lattice $\mathcal{L}_{\mathbf{x}}$ generated by the $\mathbf{x}_i$'s, given the sample vector $\mathbf{h}$ and the modulus Q.

Let $\mathcal{L}_0$ be the lattice of vectors orthogonal to $\mathbf{h}$ modulo Q:

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

Recall that the main observation from [NS99] is that 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, because from (3.1) we have

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

Then, if $\mathbf{p}_{\mathbf{u}}$ is shorter than the shortest non-zero vector orthogonal to $\boldsymbol{\alpha}$ modulo Q, the condition $\mathbf{p}_{\mathbf{u}} = 0$ must be satisfied. This implies that in such a case $\mathbf{u} \in \mathcal{L}_{\mathbf{x}}^{\perp}$.

For binary the $\mathbf{x}_i$'s, Nguyen and Stern observed that if the vector $\mathbf{u}$ is short, then $\mathbf{p_u}$ will be short, too. 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 $\bar{\mathcal{L}}_{\mathbf{x}}$.

Generally, if the coefficients of $\mathbf{x}_i$ are in $\{0, \dots, B\}$ the gap between the norm of $\mathbf{p_u}$ and $\mathbf{u}$ gains a factor at most B. The same algorithm can be hence applied, however the B factor must necessarily be taken into account, and it affects the efficacy of the attack.

Algorithm 6.2 Orthogonal lattice attack

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

The orthogonal lattice attack can also be improved in this case, as discussed in Section 4.1. For the sake of simplicity, here we first analyse the simplest version, displayed by Algorithm 6.2.

6.4.2 First step: analysis

Rigorous analysis

If the modulus Q is sufficiently large, the orthogonal lattice attack returns a basis of the lattice $\bar{\mathcal{L}}_{\mathbf{x}}$ in polynomial time for a significant fraction of the possible weight vectors $\boldsymbol{\alpha}$. Indeed, we can prove the following extension of Theorem 3.1:

Theorem 6.5. *Let $m > n$. Assume that the lattice $\mathcal{L}_{\mathbf{x}}$ has rank n. With a probability at least $1/2$ over the choice of $\boldsymbol{\alpha}$, Algorithm 6.2 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 extended subset sum density is $d = n \log(B+1)/\log Q = \mathcal{O}(1/(n \log n))$.*

Compared to HSSP, the lower bound on the modulus gains a factor $B^{(n+1)n}$, due to the enlarged range for the coefficients of the $\mathbf{x}_i$'s. In order to prove Theorem 6.5, we first extend Lemma 3.2:

Lemma 6.6. *Assume that the lattice $\mathcal{L}_{\mathbf{x}}$ has rank n. Algorithm 6.2 computes a basis of the lattice $\bar{\mathcal{L}}_{\mathbf{x}}$ in polynomial time under the condition $m > n$ and*

$$\sqrt{mn} \cdot B \cdot 2^{\frac{m}{2}} \cdot \lambda_{m-n}\left(\mathcal{L}_{\mathbf{x}}^{\perp}\right) < \lambda_1\left(\Lambda_Q^{\perp}(\boldsymbol{\alpha})\right). \tag{6.9}$$

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

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

$$\sqrt{mn}B\|\mathbf{u}\| < \lambda_1\left(\Lambda_Q^\perp(\boldsymbol{\alpha})\right). \tag{6.10}$$

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 1.14, for each $j \leq m - n$ we have

$$\|\mathbf{u}_j\| \leq 2^{\frac{m}{2}} \cdot \lambda_{m-n}(\mathcal{L}_0) \leq 2^{\frac{m}{2}} \cdot \lambda_{m-n}\left(\mathcal{L}_\mathbf{x}^\perp\right) \tag{6.11}$$

since $\mathcal{L}_\mathbf{x}^\perp$ is a sublattice of $\mathcal{L}_0$ of dimension $m - n$. Combining (6.11) with (6.10), 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 6.2 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.

□

Proof (Proof of Theorem 6.5). In order to apply Lemma 6.6, 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

$$\lambda_{m-n}\left(\mathcal{L}_\mathbf{x}^\perp\right) \leq \sqrt{\gamma_{m-n}}^{m-n} \det\left(\mathcal{L}_\mathbf{x}^\perp\right) \leq m^{m/2} \det\left(\mathcal{L}_\mathbf{x}^\perp\right). \tag{6.12}$$

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:

$$\det \mathcal{L}_\mathbf{x}^\perp \leq \det \mathcal{L}_\mathbf{x} \leq \prod_{i=1}^{n} \|\mathbf{x}_i\| \leq m^{n/2} B^n \tag{6.13}$$

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 6.6, 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 1.16, this implies that, with a 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 $\mathbf{HLCP}_{n,B}$

Our rigorous analysis shows that the hidden lattice $\bar{\mathcal{L}}_{\mathbf{x}}$ can be revealed for Q sufficiently large. For instance, if $\log Q = \mathcal{O}(n^2(\log n + \log B))$ when $m = 2n$. However, we observed that in practice smaller values for the modulus Q can be selected. One reason is that we can actually 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 can use the heuristic approximation

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

then from (6.13) we obtain:

$$\lambda_{m-n}\left(\mathcal{L}_{\mathbf{x}}^{\perp}\right) \lessapprox \sqrt{\gamma_{m-n}} B^{\frac{n}{m-n}} m^{\frac{n}{2(m-n)}}. \tag{6.14}$$

Moreover, by using the Gaussian heuristic, we expect that

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

Combining these heuristics we can approximate the terms in condition (6.9) and get

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

where $2^{\iota m}$ substitutes the $2^{m/2}$ factor corresponding to the LLL Hermite factor with $\delta = 3/4$. In practice, we always consider $\delta = 0.99$, so we denote by $2^{\iota m}$ the corresponding LLL Hermite factor.

This approximate condition is

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

This gives:

$$\log Q > \iota \cdot m \cdot n + \frac{n}{2} \log(n \cdot \gamma_{m-n}/\gamma_n) + \frac{mn}{2(m-n)} \log m + \frac{mn}{m-n} \log B \tag{6.15}$$

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

$m = 2n$ we obtain the condition:

$$\log Q > 2\iota \cdot n^2 + \frac{3n}{2}\log n + n + 2n\log B \tag{6.16}$$

For our experiments we can use $m = 2n$ and $\log Q \simeq 2\iota n^2 + n\log n + 2n\log B$ with $\iota = 0.035$.

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 $\mathcal{O}(d^4 m(d + \log\beta)\log\beta)$ without fast integer arithmetic, for vectors of Euclidean norm less than β. Algorithm 6.2 for Step 1 applies twice LLL.

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 $\mathcal{O}(m^5(m+\log Q)\log Q)$. If $m = n+k$ for some constant k, the heuristic complexity is therefore $\mathcal{O}(n^9(\log n + \log B)^2)$. If $m > c\cdot n$ for some constant c, the heuristic complexity is $\mathcal{O}(m^5 \cdot (mn + n\log B)^2)$.

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 (6.11) and (6.14), we can heuristically assume $\|\mathbf{U}\| \leq 2^{m/2} \cdot \sqrt{m} \cdot B^{\frac{n}{(m-n)}} \cdot m^{\frac{n}{2(m-n)}}$. For $m = n+k$ for some constant k, this gives $\log\|\mathbf{U}\| = \mathcal{O}(n(\log n + \log B)$, while for $m > c\cdot n$ for some constant $c > 1$, we obtain $\log\|\mathbf{U}\| = \mathcal{O}(m + \log B)$. As seen in Section 1.2, the heuristic complexity of computing the orthogonal of $\mathbf{U}$ is $\mathcal{O}(m^5(m+(m/n)\log\|\mathbf{U}\|)^2)$. For $m = n+k$, the complexity is therefore $\mathcal{O}(n^7(\log n+\log B)^2)$, while for $m > c\cdot n$, the complexity is $\mathcal{O}(m^7(m+\log B)^2/n^2)$.

We summarise the complexities of the two LLL operations in Table 6.4.1 for any B. The complexities are optimal for $m = c\cdot n$ for some constant $c > 1$, so for simplicity we take $m = 2n$.

m	$\log Q$	LLL $\mathcal{L}_0$	LLL $(\mathcal{L}_{\mathbf{x}}^{\perp})^{\perp}$
$\gg n$	$\mathcal{O}(n\cdot m + n\log B)$	$\mathcal{O}(m^5 n^2 \cdot (m+\log B)^2)$	$\mathcal{O}(m^7 \cdot (m+\log B)^2/n^2)$
$2n$	$\mathcal{O}(n^2 + n\log B)$	$\mathcal{O}(n^7(n+\log B)^2)$	$\mathcal{O}(n^5(n+\log B)^2)$
$n+1$	$\mathcal{O}(n^2(\log n + \log B))$	$\mathcal{O}(n^9(\log n+\log B)^2)$	$\mathcal{O}(n^7(\log n+\log B)^2)$

Table 6.4.1. Modulus size and time complexity of Algorithm 6.2 as a function of the parameter m.

Table 6.4.2 specifies the complexities for $B = \mathcal{O}(n)$. This is the case, for instance, of the subset sum problem, *i.e.* $B = 1$. In that case the heuristic complexity of the first step is $\mathcal{O}(n^9)$, and the density is $d = n/\log Q = \mathcal{O}(1/n)$, as in the classical subset sum problem.

Heuristic analysis $\mathbf{HLCP}^{\kappa}_{n,B}$

When $\mathbf{h}$ comes from a random instance of $\mathrm{HLCP}^{\kappa}_{n,B}$ the heuristic bounds can be slightly improved. Indeed, in such a case

m	$\log Q$	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	$\mathcal{O}(n^3)$	$\mathcal{O}(n^{16})$	$\mathcal{O}(n^{16})$
$2n$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^9)$	$\mathcal{O}(n^7)$
$n+1$	$\mathcal{O}(n^2 \log n)$	$\mathcal{O}(n^9 \log^2 n)$	$\mathcal{O}(n^7 \log^2 n)$

Table 6.4.2. Modulus size and time complexity of Algorithm 6.2 as a function of the parameter m when $B = \mathcal{O}(n)$.

$$\mathbb{E}[\|\mathbf{x}_i\|^2] = m\frac{\kappa}{n}\frac{(B+1)(2B+1)}{2} \tag{6.17}$$

Therefore, we can approximate the lattice minima using this upper bound and we get

$$\lambda_{m-n}\left(\mathcal{L}_{\mathbf{x}}^{\perp}\right) \lessapprox \sqrt{\gamma_{m-n}}\left(m\frac{\kappa}{n}\frac{(B+1)(2B+1)}{2}\right)^{\frac{n}{2(m-n)}}. \tag{6.18}$$

This implies that the update condition is

$$2^{\iota \cdot m}\sqrt{\gamma_{m-n} \cdot n}m^{\frac{m}{2(m-n)}}B \cdot \left(\frac{\kappa}{n}\frac{(B+1)(2B+1)}{2}\right)^{\frac{n}{2(m-n)}} < \sqrt{\gamma_n}Q^{1/n}$$

For $B > 1$, we have $(B+1)(2B+1)/2 < 2B^2$, so for simplicity we can require

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

Consequently (6.15) becomes

$$\begin{aligned}\log Q > \iota \cdot m \cdot n + \frac{n(m-2n)}{2(m-n)}\log(n) + \frac{n}{2}\log(\gamma_{m-n}/\gamma_n) + \\ + \frac{mn}{2(m-n)}\log m + \frac{n^2}{2(m-n)}\log(2\kappa) + \frac{mn}{2(m-n)}\log(B).\end{aligned}$$

This for $m = 2n$ gives us

$$\log Q > 2\iota \cdot n^2 + n\log(n) + \frac{n}{2}\log(\kappa) + \frac{3n}{2} + n\log(B).$$

which is smaller for $\kappa \ll n$ and $B \ll n$.

6.4.3 First step: improvements

In Section 4.1.1 we first described a method for generating a basis of n vectors of $\bar{\mathcal{L}}_{\mathbf{x}} \subset \mathbb{Z}^m$ for $m > 2n$, while applying LLL on matrices of dimension $t = 2n$ only. Recall that the values

of the complexities of the orthogonal lattice attack are almost optimal for $m = 2n$, for fixed B. This method consists of multiple applications of the orthogonal lattice attack, therefore naturally applies to the HLCP, too. Therefore, we can extend directly Proposition 4.1 and we obtain:

Corollary 6.7. *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 4.1 recovers a basis of $\bar{\mathcal{L}}_{\mathbf{x}}$ in polynomial time, assuming that Q is a prime of bitsize at least $4n^2(\log n + 1) + n(n+1)\log B$.*

Then in Section 4.1.2 we illustrated a further improvement, showing that only a single application of LLL (with the same dimension) suffices to produce the $m-n$ orthogonal vectors in $\mathcal{L}_{\mathbf{x}}^{\perp}$. Indeed, 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 to the first improvement, this strategy can be applied in this extended setting. Eventually, this implies that the heuristic complexity of the first step for solving the hidden linear combination problem is $\mathcal{O}(n^7(n + \log B)^2)$. Specifically, this is $\mathcal{O}(n^9)$ if $B = \mathcal{O}(n)$. Therefore, considering a large m affects neither the lower bound on Q nor the asymptotic time complexity.

The following is a revision of Lemma 4.6, including the updated condition on the size of vectors $\mathbf{x}_i$'s.

Lemma 6.8. *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 $1 \le i \le n$ with $\mathbf{c}_i \in \mathbb{Z}^{2n}$ and $\|\mathbf{c}_i\| \le \beta$, 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 $2n + 1 \le i \le m$ with $\mathbf{c}_i \in \mathbb{Z}^{2n}$ and $\|\mathbf{c}_i\| \le \beta$. Then if $(\gamma\beta + 1)\sqrt{mn}B \le \lambda_1\left(\Lambda_Q^{\perp}(\boldsymbol{\alpha})\right)$ where $\gamma = 1 + 4n(9/2)^n$, Algorithm 4.3 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 $1 \le i \le n$, and $m - 2n$ vectors $\mathbf{a}'_i \in \mathcal{L}_{\mathbf{x}}^{\perp}$ for $2n + 1 \le i \le m$.*

Proof. We let $\mathcal{L} \subset \mathbb{Z}^{2n}$ be the lattice generated by the LLL-reduced basis of vectors $\mathbf{a}_1, \ldots, \mathbf{a}_{2n}$. By assumption the lattice $\mathcal{L}$ contains n linearly independent vectors $\mathbf{c}_i$ with $\|\mathbf{c}_i\| \le \beta$. Therefore, we must have for all $1 \le i \le n$:

$$\|\mathbf{a}_i\| \le 2^n \cdot \lambda_n(\mathcal{L}) \le 2^n\beta \le \gamma\beta$$

This implies $\|\mathbf{a}'_i\| = \|\mathbf{a}_i\| \le \gamma\beta$. Since $(\gamma\beta + 1)\sqrt{mn}B \le \lambda_1\left(\Lambda_Q^{\perp}(\boldsymbol{\alpha})\right)$, we have $\|\mathbf{a}'_i\|\sqrt{mn}B \le \lambda_1\left(\Lambda_Q^{\perp}(\boldsymbol{\alpha})\right)$ for all $1 \le i \le n$, which corresponds to (6.10) in Lemma 6.6. Therefore, we must have $\mathbf{a}'_i \in \mathcal{L}_{\mathbf{x}}^{\perp}$ for all $1 \le i \le n$.

For $2n + 1 \le i \le m$, we consider the vector $\mathbf{t}_i = [-h_i h_1^{-1}[Q]\ 0\ \cdots\ 0] \in \mathbb{Z}^{2n}$, and the vector $\mathbf{t}'_i = [\mathbf{t}_i\ 0 \cdots 1 \cdots 0] \in \mathbb{Z}^m$ where the 1 component is at index i. We have $\mathbf{t}'_i \in \mathcal{L}_0$.

Since by assumption there exists a vector $\mathbf{c}'_i = [\,\mathbf{c}_i\ 0\ \cdots\ 1\ \cdots\ 0\,] \in \mathcal{L}^{\perp}_{\mathbf{x}} \subset \mathcal{L}_0$, we must have $\mathbf{t}'_i - \mathbf{c}'_i = [\mathbf{t}_i - \mathbf{c}_i\ 0 \cdots 0] \in \mathcal{L}_0$, and therefore $\mathbf{u} := \mathbf{t}_i - \mathbf{c}_i \in \mathcal{L}$.

Let $\mathbf{v} \in \mathcal{L}$ be the vector obtained from Babai's rounding at Step 6, when given the vector $\mathbf{t}_i$ as input. Since the lattice basis $\mathbf{a}_1, \ldots, \mathbf{a}_{2n} \in \mathbb{Z}^{2n}$ is LLL-reduced, from Theorem 4.5 we must have:

$$\|\mathbf{t}_i - \mathbf{v}\| \leq \gamma \|\mathbf{t}_i - \mathbf{u}\| = \gamma \|\mathbf{c}_i\| \leq \gamma\beta$$

where $\gamma = 1+4n(9/2)^n$. Therefore letting $\mathbf{a}'_i = [(\mathbf{t}_i - \mathbf{v})\ 0\ \ 1\ \ 0] \in \mathbb{Z}^m$, we must have $\mathbf{a}'_i \in \mathcal{L}_0$ and moreover $\|\mathbf{a}'_i\| \leq \|\mathbf{t}_i - \mathbf{v}\| + 1 \leq \gamma\beta + 1$. As previously this implies $\|\mathbf{a}'_i\|\sqrt{mn}B \leq \lambda_1\left(\Lambda^{\perp}_Q(\boldsymbol{\alpha})\right)$ and therefore we must have $\mathbf{a}'_i \in \mathcal{L}^{\perp}_{\mathbf{x}}$ for all $2n+1 \leq i \leq m$.

□

We proceed accordingly to Section 4.1.2 to estimate the complexity. Since the approximation factor γ for CVP is similar to the LLL Hermite factor, we use the same modulus size as previously, namely $\log Q \simeq 2\iota n^2 + n \cdot \log n + 2n \log B$ with $\iota = 0.035$. As in Section 3.2.1 the complexity of the first LLL reduction with L^2 is $\mathcal{O}(n^7(n + \log B)^2)$. For the Babai's rounding we must first invert a $2n \times 2n$ matrix with $\log Q$ bits of precision; this has to be done only once, and takes $\mathcal{O}(n^3 \log^2 Q) = \mathcal{O}(n^5(n + \log B)^2)$ time. Then for each Babai's rounding we need one vector matrix multiplication, with precision $\log Q$ bits. Since the vector has a single non-zero component, the complexity is $\mathcal{O}(n \log^2 Q) = \mathcal{O}(n^3(n + \log B)^2))$. If $m = \mathcal{O}(n^a B^b)$ for $a > 1$, the total complexity of size-reduction is $\mathcal{O}(n^{3+a}(n + \log B)^2)B^b)$.

6.4.4 Second step: BKZ

The goal of the second step of the algorithm is to compute the hidden vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n$ and the hidden weights $\boldsymbol{\alpha}$ from $\bar{\mathcal{L}}_{\mathbf{x}} \subset \mathbb{Z}^m$.

From the first step we have obtained an LLL-reduced basis $(\mathbf{c}_1, \ldots, \mathbf{c}_n)$ of the completed lattice $\bar{\mathcal{L}}_{\mathbf{x}} \subset \mathbb{Z}^m$. This, however, does not necessarily reveal the vectors $\mathbf{x}_i$'s. 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 $\mathbf{x}_i$'s, which are among the shortest vectors in $\mathcal{L}_{\mathbf{x}}$. Therefore, in order 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 Q, one can recover the weights α_i by solving a linear system; this is the implementation of Nguyen and Stern's second step in [NS99]. This strategy is fully described in Section 3.1.2 for the hidden subset sum problem. In the following, then we extend our analysis of the Nguyen-Stern lattice attack for $B \geq 1$, as outlined in [CG22].

Second Step $\mathbf{HLCP}_{n,B}$

In our analysis below here, for simplicity we heuristically assume that the lattice $\mathcal{L}_{\mathbf{x}}$ is complete.

Short vectors in $\mathcal{L}_{\mathbf{x}}$

The vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{Z}^m$ have coordinates distributed uniformly over the set $\{0, \ldots, B\}$. 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 $\mathbf{c}_1, \ldots, \mathbf{c}_n$ is either equal to $\pm\mathbf{x}_i$, or equal to a combination of the form $\mathbf{x}_i - \mathbf{x}_j$ for $i \neq j$.

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 $\{0, \ldots, B\}$. 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:

$$2^{\iota \cdot n} \leq \frac{1}{2}\sqrt{n\frac{B+2}{2B+1}} \simeq \frac{1}{2}\sqrt{\frac{n}{2}}. \tag{6.19}$$

According Section 1.1.3, we can achieve such a value as Hermite factor heuristically by using BKZ reduction with block-size $\beta \simeq n$. Namely, the running time of the generalised Nguyen-Stern algorithm results $\mathsf{poly}(n, \log B) \cdot 2^{\Omega(n)} = 2^{\Omega(n)} \cdot \log^{\mathcal{O}(1)} B$.

Second Step $\mathrm{HLCP}^{\kappa}_{n,B}$

A preliminary discussion about short vectors in $\bar{\mathcal{L}}_{\mathbf{x}}$ is placed in Section 6.3.1. There, we argue that if $\varsigma = \kappa/n \ll B/(B+1)$, we can expect to have several additional vectors of $\mathcal{L}_{\mathbf{x}}$ lying in $\{0,\dots,B\}^m$. We now then consider the case $\varsigma = \kappa/n \gtrsim B/(B+1)$.

For any $i \neq j \in 0,\dots,n$ we have $\mathbb{E}[\|\mathbf{x}_i\|^2] = m\frac{\kappa}{n}\frac{(B+1)(2B+1)}{2}$ and

$$\mathbb{E}[\|\mathbf{x}_i - \mathbf{x}_j\|^2] = m \cdot 2\operatorname{Var}(x_{k,l}) = m\frac{\kappa(B+1)}{2n}\left(\frac{2B+1}{3} - \frac{\kappa}{n}\frac{(B+1)}{2}\right).$$

This suggests that for balanced $\mathrm{HLCP}^{\kappa}_{n,B}$ we can expect a behaviour very similar to $\mathrm{HLCP}_{n,B}$.

By combining BKZ with an extended greedy strategy as described for HSSP, we can try to recover the target vector in these cases, but when the problem is not balanced we can not rely on the presence of a gap. While, when the problem is balanced, our heuristic analysis of $\mathrm{HLCP}_{n,B}$ suggests that anyway the algorithm remains exponential.

6.4.5 Practical Experiments

We performed experiments for the extended Nguyen-Stern attack for $B = 10$. Table 6.4.3 is a collection of timings for $\mathrm{HLCP}_{n,10}$ for various n, while Table 6.4.4 and Table 6.4.5 for $\mathrm{HLCP}^{\kappa}_{n,10}$ with $\kappa/n \simeq 10/11$ and $\kappa/n \simeq 1/2$, respectively. Notice that when $\kappa/n \simeq 10/11$, we expect random instances of $\mathrm{HLCP}^{\kappa}_{n,10}$ to be very similar to $\mathrm{HLCP}_{n,10}$. As for the HSSP, *i.e.* $B = 1$, we face here an exponential barrier in the second step for $n > 170$, too.

			Step 1		Step 2		
n	m	$\log Q$	LLL $\mathcal{L}_0$	LLL $\mathcal{L}^{\perp}_{\mathbf{x}}$	Reduction		Total
70	140	1237	5.2 s	3.0 s	BKZ-10	0.7 s	9.0 s
90	180	1749	16.8 s	8.9 s	BKZ-10	1.6 s	27.5 s
110	220	2323	42.1 s	21.6 s	BKZ-10	4.4 s	68.7 s
130	260	2959	101.0 s	45.2 s	BKZ-20	14.3 s	161.3 s
150	300	3655	4 min	82.0 s	BKZ-30	3 min	8 min
170	340	4412	24 min	4 min	BKZ-30	187 min	214 min

Table 6.4.3. Running time of the Nguyen-Stern attack for $B = 10$.

				Step 1		Step 2		
n	κ	m	$\log Q$	LLL $\mathcal{L}_0$	LLL $\mathcal{L}_{\mathbf{x}}^{\perp}$	Reduction		Total
70	63	140	1237	6.1 s	3.6 s	LLL	0.7 s	10.7 s
90	81	180	1749	20.4 s	10.8 s	BKZ-10	2.1 s	33.6 s
110	100	220	2323	53.5 s	25.5 s	BKZ-10	5.3 s	84.9 s
130	118	260	2959	127.0 s	53.2 s	BKZ-20	18.4 s	3 min
150	136	300	3655	4 min	100.8 s	BKZ-30	172.9 s	9 min
170	154	340	4412	19 min	3 min	BKZ-30	137 min	159 min

Table 6.4.4. Running time of the Nguyen-Stern attack for $B = 10$ and $\kappa/n \simeq 10/11$.

				Step 1		Step 2		
n	κ	m	$\log Q$	LLL $\mathcal{L}_0$	LLL $\mathcal{L}_{\mathbf{x}}^{\perp}$	Reduction		Total
70	35	140	1237	6.8 s	4.0 s	LLL	0.8 s	11.8 s
90	45	180	1749	20.7 s	11.0 s	BKZ-10	2.1 s	34.1 s
110	55	220	2323	52.5 s	27.2 s	BKZ-10	5.8 s	86.2 s
130	65	260	2959	131.1 s	56.4 s	BKZ-20	18.1 s	3 min
150	75	300	3655	4 min	110.6 s	BKZ-30	139.3 s	8 min
170	85	340	4412	20 min	3 min	BKZ-30	436 min	458 min

Table 6.4.5. Running time of the Nguyen-Stern attack for $B = 10$ and $\kappa/n \simeq 1/2$.

6.5 Our algorithm based on statistical learning for HLCP

The hidden linear combination problem is an extension of the hidden subset sum problem where the vector $\mathbf{x}_i$'s have components lying in the range $\{0, \ldots, B\}$. We denote by $\mathbf{X} \in \mathbb{Z}^{n \times m}$ the matrix of row vectors $\mathbf{x}_i \in \mathbb{Z}^m$. By using the (improved) Nguyen-Stern orthogonal lattice attack, we can obtain[1] a matrix $\mathbf{C} \in \mathbb{Q}^{n \times m}$ such that

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

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

We can repeat then the same observations of Chapter 5. Recall that our statistical learning approach for solving the hidden subset sum problem ($B = 1$) relies on the idea of considering the m columns of the output of the orthogonal lattice attack as samples from the *discrete parallelepiped* associated to $\mathbf{V}$:

[1] See Section 5.3 for a detailed explanation.

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

where $\mathbf{v}_1, \ldots, \mathbf{v}_n \in \mathbb{Q}^n$ are the columns of $\mathbf{V}$. See Figure 5.1. Similarly, we can interpret more generally the columns of $\mathbf{C}$ as samples from

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

Thus, we want to recover the matrix $\mathbf{V}$ from those samples, by using a similar statistical technique. Indeed, given $\mathbf{V}$ one can compute the $\mathbf{x}_i$'s with $\mathbf{X} = \mathbf{V}^{-1}\mathbf{C}$.

As displayed also by Figure 5.2, Algorithm 5.1 consists of combining samples uniform over $\mathcal{P}_{\{0,1\}}(\mathbf{V})$ to obtain samples from $\mathcal{P}_{[-1,1]}(\mathbf{V})$, so that we can apply the Nguyen-Regev algorithm and get an approximation of $\mathbf{V}$; the algorithm concludes with a rounding procedure. This algorithm is highly impractical, since it requires a very large sample size m. Therefore, in this section, we present a heuristic variant working with fewer samples. Indeed, the variant we propose here processes directly samples from the uniform distribution over the discrete parallelepiped, avoiding the combination step. Moreover, we show that this heuristic variant works for $B \geq 1$, too.

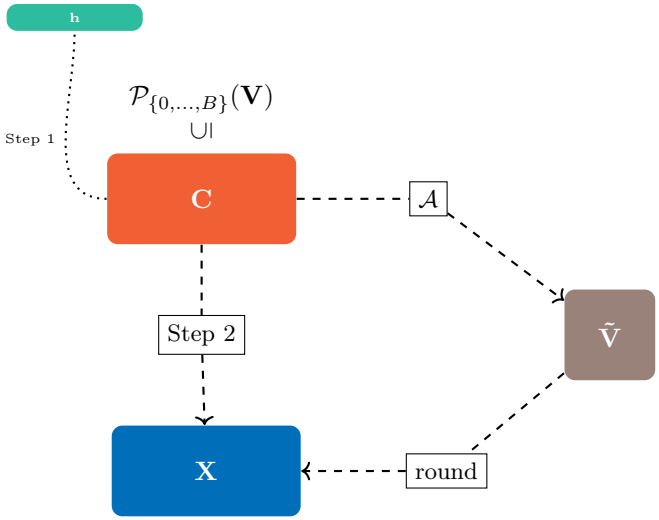


Fig. 6.1. Our heuristic algorithm for HLCP based on statistical learning.

6.5.1 Solving $\mathrm{HLCP}_{n,B}$

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 $\mathbf{L}^{-1}$.

The overall strategy of this algorithm is the same as that one of the FastICA algorithm by Aapo Hyvärinen and Erkki Oja [HO97], *i.e.* an algorithm for solving the *signal source separation problem*. 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 $\mathcal{U}(\mathcal{P}_{[-1,1]}(\mathbf{A}))$. Studying these algorithms, one can notice that the choice of parameters in [NR09] makes Nguyen-Regev's gradient descent coincide with the fixed-point algorithm used in FastICA, when the distribution of the sources is $\mathcal{U}([-1,1])$ and with kurtosis chosen as cost function. Therefore, our idea is to directly use a generic statistical method derived from FastICA to solve the hidden linear combination problem. Our algorithm is summarised by Algorithm 6.3, where we denote Algorithm 4.1 by $\mathsf{BlocksOrthoLat}$. We recall that as the orthogonal lattice attack, this can be directly adapted for $B > 1$. See Section 6.4.3.

Algorithm 6.3 Statistical attack for HLCP

Input: $\mathbf{h}, Q, n$

Output: the vectors $\mathbf{x}_i$.

1: $[\mathbf{C}_0, \mathbf{C}_1, \mathbf{C}'_2, \cdots, \mathbf{C}'_b] \leftarrow \mathsf{BlocksOrthoLat}(\mathbf{h}, n, m, B)$
2: Use statistical learning to compute an approximation $\tilde{\mathbf{V}}$ of $\mathbf{V}$.
3: Recover $\mathbf{X}$ from the rounding of $\tilde{\mathbf{X}} = \tilde{\mathbf{V}}^{-1}\mathbf{C}$.
4: **return** $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$

We use here the FastICA for performing statistical learning as a black-box, and we refer to [HO97, HO00, HKO04] for details.

As in the Nguyen-Regev algorithm, the statistical learning algorithm (FastICA) 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.

The complexity of the underlying statistical learning method is polynomial in the size of the input sample and the size of such vectors' coefficients [HKO04]. Hence, we can conclude that the heuristic complexity of our statistical approach for solving $\mathrm{HLCP}_{n,B}$ is $\mathsf{poly}(n, B)$. Recall that the Nguyen-Stern attack from Section 6.4 has complexity exponential in n but polynomial in $\log B$.

We observed that in practice this algorithm is quite efficient. In particular, we can recover the binary vectors of a random HSSP_n in only a few seconds for $n < 250$ for $m \approx n^2$. Moreover, in such a case the space complexity of the second step is $\mathcal{O}(n^3)$ only, instead of $\mathcal{O}(n^4)$ as in the multivariate attack. We collected practical experiments in Section 6.5.2.

Remark 6.9. In Chapter 5 we used the Nguyen-Regev algorithm to prove that there exists an algorithm solving HSSP_n in polynomial time (Theorem 5.4). Specifically, we showed that combining a polynomial-size sample of $\mathcal{P}_{\{0,1\}}(\mathbf{V})$ one can obtain a sample of $\mathcal{P}_{[-1,1]}(\mathbf{V})$, which is sufficiently large for successfully applying the method described in [NR09]. For getting a theoretical result for $\mathrm{HLCP}_{n,B}$ similar to Theorem 5.4, we should then prove that combining enough elements sampled uniformly from $\mathcal{P}_{\{0,\dots,B\}}(\mathbf{V})$ we can obtain sufficiently many random elements of $\mathcal{P}_{[-1,1]}(\mathbf{V})$. This is however less straightforward. Notice that, for instance, when $B > 1$ we can not center the distribution only by subtracting a vector as in equation (5.1); we should instead use the mean vector of $\mathcal{P}_{\{0,\dots,B\}}(\mathbf{V})$, which however depends on the unknown distribution $\mathbf{V}$. A possible solution is to consider a sample estimation of the mean, but this requires introducing further conditions. The behaviour of our heuristic variant suggests that a lower bound for m obtained using this *modus operandi* would not be anyway tight. Therefore, we decided to not extend Theorem 5.4 here.

Remark 6.10. Any vector belonging to one of the previously considered parallelepipeds can be written as $\mathbf{V} \cdot \mathbf{x}$, where $\mathbf{x}$ belongs either to $[-1,1]^n$, $\{0,1\}^n$ or $\{0,\dots,B\}^n$. Both the Nguyen-Regev algorithm and FastICA are based on the assumption the components of $\mathbf{x}$ are sampled independently[2]. For this reason we can not expect our algorithm to succeed for instances of $\mathrm{HLCP}^{\kappa}_{n,B}$, since by constructions the columns of $\mathbf{X}$ do not satisfy this hypothesis in such a case. See Definition 6.4. However, we have seen that for some combinations of n, κ and B Algorithm 6.3 actually works. We discuss this fact in Section 6.5.3

Remark 6.11. When discussing the differences between the original setting for which the Nguyen-Regev algorithm was developed and the one we are analysing in this thesis, we have observed in our case the entries of $\mathbf{V}$ are not polynomially bounded in absolute value. See Chapter 5. Although this motivates Step 3 of Algorithm 6.3, in practice we can often round $\tilde{\mathbf{V}}$, directly. Indeed, there is often gap between the theoretical and practical bounds for LLL, and the size of the coefficients of $\mathbf{V}$ indirectly depends on the quality of the reduction applied to the output of the orthogonal step. Intuitively, if the basis $\mathbf{C}$ is composed of vectors with small coefficients we can expect those of $\mathbf{V}$ to be small, too. Then, in such a case we can apply the rounding to $\tilde{\mathbf{V}}$, directly.

6.5.2 Practical experiments for $\mathrm{HLCP}_{n,B}$

We described a heuristic attack based on statistical learning. We claimed that this algorithm solves random instances for HSSP_n and $\mathrm{HLCP}_{n,B}$ in $\mathsf{poly}(n, B)$.

[2] ICA is the acronym of independent component analysis.

We provide in Table 6.5.1 the results of practical experiments running our statistical attack for $B = 1$, in Table 6.5.2 for $B = 2$ and in Table 6.5.3 for $B = 10$. We have used the implementation of the 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 for HSSP_n, the second step takes 45 minutes. Our implementation is available at `https://github.com/agnesegini/solving_hssp`.

For $B = 10$, in our experiments the statistical attack is less efficient than the Nguyen-Stern algorithm, as one must generate a large number m of samples in the first step. However the attack scales polynomially with n; therefore, we expect the statistical approach to eventually outperform the Nguyen-Stern attack for larger values of n. The content of this section partially appeared in [CG22].

n	m	$\log Q$	Step 1: LLL	Step 2: FastICA	Total
70	2590	772	9.9 s	10.7 s	20.9 s
90	4230	1151	21.7 s	40.4 s	62.9 s
110	6270	1592	49.6 s	141.4 s	3 min
130	16900	2095	129.7 s	54.2 s	3 min
150	22500	2659	4 min	72.5 s	6 min
170	28900	3282	8 min	110.7 s	10 min
190	36100	3965	26 min	156.5 s	29 min
220	48400	5099	96 min	4 min	99 min
250	62500	7362	230 min	76 s	233 min

Table 6.5.1. Running time of our statistical attack for HSSP_n.

n	m	$\log Q$	Step 1: LLL	Step 2: FastICA	Total
70	9800	912	22.6 s	28.4 s	52.2 s
90	16200	1331	70.5 s	33.3 s	107.0 s
110	24200	1812	153.3 s	65.5 s	4 min
130	33800	2355	5 min	93.2 s	7 min
150	45000	2959	11 min	137.1 s	13 min
170	57800	3622	14 min	4 min	18 min
190	72200	4345	34 min	4 min	38 min

Table 6.5.2. Running time of our statistical attack for $\mathrm{HLCP}_{n,2}$.

n	m	$\log Q$	Step 1: LLL	Step 2: FastICA	Total
70	19600	1237	36.0 s	27.1 s	64.6 s
90	32400	1749	98.8 s	49.6 s	152.2 s
110	48400	2323	4 min	68.4 s	5 min
130	67600	2959	7 min	142.0 s	10 min
150	90000	3655	12 min	4 min	17 min

Table 6.5.3. Running time of our statistical attack for $\mathrm{HLCP}_{n,10}$.

6.5.3 Application to $\mathrm{HLCP}^{\kappa}_{n,B}$

The statistical learning algorithm we describe in Section 6.5.1 relies on the assumption that the vectors $\mathbf{x}_i$'s are statistically independent. Specifically, it requires the coefficients columns of $\mathbf{X}$ to be independently distributed. By construction the instances of $\mathrm{HLCP}^{\kappa}_{n,B}$ do not satisfy this property, since the columns of $\mathbf{X}$ must have fixed Hamming weight. See Definition 6.4. However, running the algorithm over several instances, we noticed that for some combinations of n, κ and B the algorithm actually retrieves the $\mathbf{x}_i$'s for $B > 1$. We provide now the intuition of a possible reason.

Consider a column $\hat{\mathbf{x}}_i$ of $\mathbf{X}$, we have that the non-zero coordinates of $\hat{\mathbf{x}}_i$ are distributed randomly uniformly over $\{1, \ldots, B\}$. This implies that if ς is sufficiently close $B/(B+1)$ the distribution of $\hat{\mathbf{x}}_i$ is close to the uniform distribution over $\{0, \ldots, B\}^n$. Similarly, if ς tends to 1, $\hat{\mathbf{x}}_i$'s distribution tends to the uniform distribution over $\{1, \ldots, B\}^n$. Given the characteristics exploited by the algorithm[3], this is in fact sufficient to succeed. Tables 6.5.4, 6.5.5, 6.5.6 and 6.5.7 contain examples of running-time for some different tuples of (n, κ, B). Notice that when κ/n gets closer either to $B/(B+1)$ or 1, we have that more vectors are recovered and that the second step becomes faster.

n	κ	m	Step 1	Step 2	recovered $\mathbf{x}_i$	tot
90	45	16200	41.3 s	13.4 s	90	54.7 s
90	55	16200	38.5 s	11.7 s	90	50.2 s
90	65	16200	42.2 s	11.9 s	90	54.1 s
90	75	16200	38.9 s	14.9 s	90	53.8 s
90	85	16200	41.9 s	40.5 s	90	82.4 s
90	90	16200	34.3 s	9.9 s	90	44.3 s

Table 6.5.4. Running time of our heuristic statistical attack applied to $\mathrm{HLCP}^{\kappa}_{90,2}$ for various κ.

[3] The correctness and convergence of the FastICA algorithm depends on the fact that we can reduce the given vectors to a sample having a base distribution with covariance matrix close to the identity matrix. Since we did not discuss the algorithm here, we refer to [HKO04] for more details.

n	κ	m	Step 1	Step 2	recovered $\mathbf{x}_i$	tot
120	60	28800	138.7 s	29.8 s	120	168.5 s
120	70	28800	142.9 s	25.9 s	120	168.8 s
120	80	28800	144.7 s	29.4 s	120	174.1 s
120	90	28800	168.9 s	30.9 s	120	3 min
120	100	28800	150.9 s	30.7 s	120	3 min
120	110	28800	135.6 s	93.1 s	0	4 min
120	120	28800	154.9 s	29.5 s	120	3 min

Table 6.5.5. Running time of our heuristic statistical attack applied to $\mathrm{HLCP}^{\kappa}_{120,2}$ for various κ.

n	κ	m	Step 1	Step 2	recovered $\mathbf{x}_i$	tot
140	70	39200	6 min	50.5 s	140	7 min
140	80	39200	7 min	49.5 s	140	8 min
140	90	39200	6 min	53.4 s	140	7 min
140	100	39200	7 min	48.1 s	140	7 min
140	110	39200	5 min	46.3 s	140	6 min
140	120	39200	5 min	52.7 s	6	6 min
140	130	39200	5 min	149.1 s	0	7 min
140	140	39200	4 min	37.6 s	140	5 min

Table 6.5.6. Running time of our heuristic statistical attack applied to $\mathrm{HLCP}^{\kappa}_{140,2}$ for various κ.

n	κ	m	Step 1	Step 2	recovered $\mathbf{x}_i$	tot
120	50	28800	152.0 s	92.5 s	0	4 min
120	60	28800	165.9 s	40.5 s	120	3 min
120	70	28800	3 min	33.9 s	120	4 min
120	80	28800	163.6 s	27.5 s	120	3 min
120	90	28800	160.2 s	30.4 s	120	3 min
120	100	28800	155.1 s	31.4 s	120	3 min
120	110	28800	153.8 s	33.1 s	120	3 min
120	120	28800	147.2 s	25.9 s	120	173.1 s

Table 6.5.7. Running time of our heuristic statistical attack applied to $\mathrm{HLCP}^{\kappa}_{120,3}$ for various κ.

6.6 Conclusions and remarks

In this chapter we studied the hardness of the affine hidden subset sum problem and the hidden linear combination problem, two variants of the hidden linear combination problem having application in cryptanalysis. For the first variant we showed that the orthogonal lattice attack can be extended and applied to solve this variant, too. This implies that all the algorithms

presented in the previous chapters for retrieving the binary vectors are still applicable. Specifically, we analysed the affine orthogonal lattice attack, providing rigorous conditions for its success. Differently from our analysis of HSSP we did not make this condition explicit, nor compute heuristic bounds. The reason is that there are several configurations that may occur, depending on the context in which the problem is placed. We recall that the BPV generator can be embedded in different primitives. Therefore, studying some of these configurations may be a future research direction.

The second variant of HSSP that we analysed is the hidden linear combination problem. First, we studied the extension of the Nguyen-Stern lattice-based attack to this problem, and then we described a heuristic variant of the statistical learning attack, working in polynomial time. For both these attacks we provided the result of our experiments. We observed that the statistical algorithm is quite efficient in practice, *e.g.* it recovers the binary vectors of HSSP_n in a few seconds only. Moreover, we saw that this method also works when the Hamming weight is fixed, for certain combinations of n, κ, B if $B > 1$.

In Chapter 5 we showed that there exists a provable polynomial-time algorithm for solving HSSP_n, via statistical learning. The variant of that algorithm discussed in this chapter for HLCP is heuristic, only. In fact, we did not show any precise bounds on its complexity. See Remark 6.9. Furthermore, it remains still unclear if we can adapt this technique to properly solve $\mathrm{HLCP}^{\kappa}_{n,B}$. Therefore, we leave these points as open questions for future works, and we conclude with Tables 6.6.1 and Tables 6.6.2, summarising the current state of the art about the algorithms for solving HSSP and HLCP, also including our contributions.

		complexity	status
HSSP_n $(B = 1)$	Nguyen-Stern [NS99]	$2^{\Omega(n)}$	heuristic
	Coron-Gini [CG20a]	$\mathcal{O}(n^9)$	heuristic
	Statistical attack [CG22]	$\mathsf{poly}(n)$	proven
$\mathrm{HLCP}_{n,B}$	Nguyen-Stern [NS99]	$2^{\Omega(n)} \cdot \log^{\mathcal{O}(1)} B$	heuristic
	Statistical attack [CG22]	$\mathsf{poly}(n, B)$	heuristic

Table 6.6.1. Summary of the algorithmic complexities for HSSP_n and $\mathrm{HLCP}_{n,B}$.

		complexity	status
HSSP_n^κ $(B=1)$	Nguyen-Stern [NS99]	$2^{\Omega(n)}$	heuristic
	Coron-Gini [CG20a]	$\mathcal{O}(n^9)$	heuristic
$\mathrm{HLCP}_{n,B}^\kappa$ $(B>1)$	Nguyen-Stern [NS99]	$2^{\Omega(n)} \cdot \log^{\mathcal{O}(1)} B$	heuristic
	Statistical attack	$\mathsf{poly}(n, B)$	heuristic

Table 6.6.2. Summary of the algorithmic complexities for HSSP_n^κ and $\mathrm{HLCP}_{n,B}^\kappa$ for $\varsigma \approx B/(B+1)$.

List of Figures

List of Tables

Acknowledgements

I wish to thank Prof. Jean-Sebastien Coron. Thank you for giving me the opportunity to work with you, always supporting me, listening to me, giving space to my ideas, being very kind and patient with me and sharing with me your passion for research.
I wish to thank the other two members of my CET Prof. Gabor Wiese and Prof. Frederik Vercauteren. Thank you for being with me along this journey. I was always looking forward to our meetings and to your questions and remarks, since every time you helped me to take a step forward in my work.
I wish to thank the non-supervising jury members Prof. Damien Stehlé and Prof. Phong Nguyen. Thank you for the interesting discussions and your valuable feedback.
I wish to thank my colleagues and friends I met through UL. You made these almost 4 years extremely pleasant. You both made the difficult times easier and the good times more enjoyable. I will always treasure the memories we made together.
I wish to thank my friend *di sempre*. You are my safe place.
I wish to thank Carlo. Thank you for everything, especially believing in me even more than I did.
I wish to thank my family. You taught me that "somehow we can solve the problem, always!" and, maybe, I took this too literally... Seriously, thank you for always supporting me, trusting me and loving me all my life.

Sincerely,

Agnese

References

[ABC+17] Giuseppe Ateniese, Giuseppe Bianchi, Angelo T. Capossele, Chiara Petrioli, and Dora Spenza. Low-cost standard signatures for energy-harvesting wireless sensor networks. *ACM Trans. Embed. Comput. Syst.*, 16(3), apr 2017.

[ACFP14] Martin R. Albrecht, Carlos Cid, Jean-Charles Faugère, and Ludovic Perret. Algebraic algorithms for lwe. Cryptology ePrint Archive, Report 2014/1018, 2014. `https://ia.cr/2014/1018`.

[AD97] Miklós Ajtai and Cynthia Dwork. A public-key cryptosystem with worst-case/average-case equivalence. *Electron. Colloquium Comput. Complex.*, 3, 1997.

[AG11] Sanjeev Arora and Rong Ge. New algorithms for learning in presence of errors. In Luca Aceto, Monika Henzinger, and Jiří Sgall, editors, *Automata, Languages and Programming*, pages 403–415, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[AJPS18] Divesh Aggarwal, Antoine Joux, Anupam Prakash, and Miklos Santha. A new public-key cryptosystem via Mersenne numbers. In *Advances in Cryptology – CRYPTO 2018*, volume 10993 of *Lecture Notes in Computer Science*, pages 459–482. Springer, 2018.

[Ajt96] Miklós Ajtai. Generating hard instances of lattice problems. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 99–108, 1996.

[ALNSD20] Divesh Aggarwal, Jianwei Li, Phong Q. Nguyen, and Noah Stephens-Davidowitz. Slide reduction, revisited—filling the gaps in svp approximation. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020*, pages 274–295, Cham, 2020. Springer International Publishing.

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

[BCGN17] Marc Beunardeau, Aisling Connolly, Rémi Géraud, and David Naccache. On the hardness of the Mersenne low Hamming ratio assumption. In *Progress in*

Cryptology - LATINCRYPT 2017, 2017. Available at `https://eprint.iacr.org/2017/522`.

[BCJ11] Anja Becker, Jean-Sébastien Coron, and Antoine Joux. Improved generic algorithms for hard knapsacks. In Kenneth G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, pages 364–385, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[BFP12] Luk Bettale, Jean-Charles Faugère, and Ludovic Perret. Solving polynomial systems over finite fields: Improved analysis of the hybrid approach. In *Proceedings of the 37th International Symposium on Symbolic and Algebraic Computation*, ISSAC '12, page 67–74, New York, NY, USA, 2012. Association for Computing Machinery.

[BFSY] Magali Bardet, Jean-Charles Faugère, Bruno Salvy, and B-Y. Yang. Asymptotic behaviour of the degree of regularity of semi-regular polynomial systems. In *IN MEGA'05, 2005. EIGHTH INTERNATIONAL SYMPOSIUM ON EFFECTIVE METHODS IN ALGEBRAIC GEOMETRY.*

[BNNT11] Eric Brier, David Naccache, Phong Q. Nguyen, and Mehdi Tibouchi. Modulus fault attacks against RSA-CRT signatures. In *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, pages 192–206, 2011.

[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.

[Cas97] J. Cassels. *An Introduction to the Geometry of Numbers*. Classics in Mathematics. Springer-Verlag, Berlin, 1997. Corrected reprint of the 1971 edition.

[CG20a] 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`.

[CG20b] Jean-Sébastien Coron and Agnese Gini. Improved cryptanalysis of the AJPS mersenne based cryptosystem. *Journal of Mathematical Cryptology*, 14(1):218–223, 2020.

[CG22] Jean-Sébastien Coron and Agnese Gini. Provably solving the hidden subset sum problem via statistical learning. *Mathematical Cryptology*, 1(2):70–84, Mar. 2022.

[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.

[CKPS00] Nicolas Courtois, Alexander Klimov, Jacques Patarin, and Adi Shamir. Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In *Proceedings of EUROCRYPT 2000*, pages 392–407, 2000.

[CLO98] David Cox, John Little, and Donal O'Shea. Solving polynomial equations. In *Using Algebraic Geometry*, pages 24–70. Springer New York, 1998.

[CLT13] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Practical multilinear maps over the integers. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, pages 476–493, 2013.

[CLV21] Celine Chevalier, Fabien Laguillaumie, and Damien Vergnaud. Privately Outsourcing Exponentiation to a Single Server: Cryptanalysis and Optimal Constructions. *Algorithmica*, 83(1):72–115, January 2021.

[CMR17] Claude Carlet, Pierrick Méaux, and Yann Rotella. Boolean functions with restricted input and their robustness; application to the FLIP cipher. *IACR Trans. Symmetric Cryptol.*, 2017(3), 2017.

[CMT01] Jean-Sébastien Coron, David M'Raïhi, and Christophe Tymen. Fast generation of pairs (k, [k]P) for Koblitz elliptic curves. In Serge Vaudenay and Amr M. Youssef, editors, *Selected Areas in Cryptography*, pages 151–164, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[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.

[CN19] Jean-Sébastien Coron and Luca Notarnicola. Cryptanalysis of CLT13 multilinear maps with independent slots. In *Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part II*, pages 356–385, 2019.

[CNT10] Jean-Sébastien Coron, David Naccache, and Mehdi Tibouchi. Fault attacks against emv signatures. In *Topics in Cryptology - CT-RSA 2010, The Cryptographers' Track at the RSA Conference 2010, San Francisco, CA, USA, March 1-5, 2010. Proceedings*, pages 208–220, 2010.

[Cou01] Nicolas T. Courtois. The security of hidden field equations (HFE). In David Naccache, editor, *Topics in Cryptology — CT-RSA 2001*, pages 266–281, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[CP19] Jean-Sébastien Coron and Hilder V. L. Pereira. On Kilian's randomization of multilinear map encodings. In *Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part II*, pages 325–355, 2019.

[CR88] Benny Chor and Ronald L. Rivest. A knapsack-type public key cryptosystem based on arithmetic in finite fields. *IEEE Transactions on Information Theory*, 34(5):901–909, 1988.

[CSV18] Jingwei Chen, Damien Stehlé, and Gilles Villard. Computing an LLL-reduced basis of the orthogonal lattice. In *Proceedings of the 2018 ACM International Symposium on Symbolic and Algebraic Computation*, ISSAC '18, page 127–133, New York, NY, USA, 2018. Association for Computing Machinery.

[dBDJdW18] Koen de Boer, Léo Ducas, Stacey Jeffery, and Ronald de Wolf. Attacks on the AJPS Mersenne-based cryptosystem. In *Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018, Fort Lauderdale, FL, USA, April 9-11, 2018, Proceedings*, pages 101–120, 2018.

[DS05] Jintai Ding and Dieter Schmidt. Rainbow, a new multivariable polynomial signature scheme. In John Ioannidis, Angelos Keromytis, and Moti Yung, editors, *Applied Cryptography and Network Security*, pages 164–175, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[Elg85] Taher Elgamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[Fau99] Jean-Charles Faugére. A new efficient algorithm for computing Gröbner bases (F4). *Journal of Pure and Applied Algebra*, 139(1):61–88, 1999.

[Fau02] Jean Charles Faugère. A new efficient algorithm for computing Gröbner bases without reduction to zero (F5). In *Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation*, ISSAC '02, page 75–83, New York, NY, USA, 2002. Association for Computing Machinery.

[FGUO+11] Jean-Charles Faugère, Valérie Gauthier-Umanã, Ayoub Otmani, Ludovic Perret, and Jean-Pierre Tillich. A distinguisher for high rate McEliece cryptosystems. In *2011 IEEE Information Theory Workshop*, pages 282–286, 2011.

[FJ03] Jean-Charles Faugère and Antoine Joux. Algebraic cryptanalysis of hidden field equation (HFE) cryptosystems using gröbner bases. In *CRYPTO*, 2003.

[FLLT15] Pierre-Alain Fouque, Moon Sung Lee, Tancrède Lepoint, and Mehdi Tibouchi. Cryptanalysis of the co-acd assumption. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, pages 561–580, 2015.

[FOP+16] Jean-Charles Faugère, Ayoub Otmani, Ludovic Perret, Frédéric De Portzamparc, and Jean-Pierre Tillich. Structural cryptanalysis of mceliece schemes with compact keys. *Designs, Codes and Cryptography*, 79(1):87–112, 2016.

[fpl16] *The FPLLL development team. fplll, a lattice reduction library*, 2016. Available at `https://github.com/fplll/fplll`.

[Fri86] Alan M. Frieze. On the lagarias-odlyzko algorithm for the subset sum problem. *SIAM Journal on Computing*, 15(2):536–539, 1986.

[GJ90] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., USA, 1990.

[GM22] Agnese Gini and Pierrick Méaux. On the weightwise nonlinearity of weightwise perfectly balanced functions. Cryptology ePrint Archive, Report 2022/408, 2022. `https://ia.cr/2022/408`.

[GN08] Nicolas Gama and Phong Q. Nguyen. Predicting lattice reduction. In Nigel Smart, editor, *Advances in Cryptology – EUROCRYPT 2008*, pages 31–51, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[HGJ10] Nick Howgrave-Graham and Antoine Joux. New generic algorithms for hard knapsacks. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, pages 235–256, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[HILL99] Johan HÅstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM J. Comput.*, 28(4):1364–1396, mar 1999.

[HKO04] Aapo Hyvarinen, Juha Karhunen, and Erkki Oja. *Independent Component Analysis*. Adaptive and Cognitive Dynamic Systems: Signal Processing, Learning, Communications and Control. Wiley, 2004.

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

[HO00] Aapo Hyvärinen and Erkki Oja. Independent component analysis: Algorithms and applications. *Neural Networks*, 13(4-5):411–430, 2000.

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

[IN96] Russell Impagliazzo and Moni Naor. Efficient cryptographic schemes provably as secure as subset sum. *Journal of Cryptology*, 9:199–216, 1996.

[KRS+11] Lakshmi Kuppusamy, Jothi Rangasamy, Douglas Stebila, Colin Boyd, and Juan Gonzalez Nieto. Towards a provably secure Dos-resilient key exchange protocol with perfect forward secrecy. In *Proceedings of the 12th International Conference on Cryptology in India*, INDOCRYPT'11, page 379–398, Berlin, Heidelberg, 2011. Springer-Verlag.

[KRS+12] Lakshmi Kuppusamy, Jothi Rangasamy, Douglas Stebila, Colin Boyd, and Juan González Nieto. Practical client puzzles in the standard model. In *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '12, page 42–43, New York, NY, USA, 2012. Association for Computing Machinery.

[KS98] Aviad Kipnis and Adi Shamir. Cryptanalysis of the oil and vinegar signature scheme. In Hugo Krawczyk, editor, *Advances in Cryptology — CRYPTO '98*, pages 257–266, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[KS99] Aviad Kipnis and Adi Shamir. Cryptanalysis of the HFE public key cryptosystem by relinearization. In Michael Wiener, editor, *Advances in Cryptology — CRYPTO' 99*, pages 19–30, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[Laz83] Daniel Lazard. Gröbner-bases, gaussian elimination and resolution of systems of algebraic equations. In *Proceedings of the European Computer Algebra Conference on Computer Algebra*, EUROCAL '83, page 146–156, Berlin, Heidelberg, 1983. Springer-Verlag.

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

[LN20] Jianwei Li and Phong Q. Nguyen. A complete analysis of the bkz lattice reduction algorithm. Cryptology ePrint Archive, Paper 2020/1237, 2020. `https://eprint.iacr.org/2020/1237`.

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

[LPS10] Vadim Lyubashevsky, Adriana Palacio, and Gil Segev. Public-key cryptographic primitives provably as secure as subset sum. In *Proceedings of the 7th International Conference on Theory of Cryptography*, TCC'10, page 382–400, Berlin, Heidelberg, 2010. Springer-Verlag.

[LT15] Tancrède Lepoint and Mehdi Tibouchi. Cryptanalysis of a (somewhat) additively homomorphic encryption scheme used in PIR. In *Financial Cryptography and Data Security - FC 2015 International Workshops, BITCOIN, WAHC, and Wearable, San Juan, Puerto Rico, January 30, 2015, Revised Selected Papers*, pages 184–193, 2015.

[LYZ+20] Hongjun Li, Jia Yu, Hanlin Zhang, Ming Yang, and Huaqun Wang. Privacy-preserving and distributed algorithms for modular exponentiation in IoT with edge computing assistance. *IEEE Internet of Things Journal*, 7(9):8769–8779, 2020.

[Mar13] Jacques Martinet. *Perfect lattices in Euclidean spaces*, volume 327. Springer, 2013.

[MH78] Ralph C. Merkle and Martin E. Hellman. Hiding information and signatures in trapdoor knapsacks. *IEEE Trans. Inf. Theory*, 24:525–530, 1978.

[Min97] H. Minkowski. *Geometrie der Zahlen.* Teubner, Berlin, 1997. Corrected reprint of the 1971 edition.

[MJSC16] Pierrick Méaux, Anthony Journault, François-Xavier Standaert, and Claude Carlet. Towards stream ciphers for efficient FHE with low-noise ciphertexts. In *Proceedings, Part I, of the 35th Annual International Conference on Advances in Cryptology — EUROCRYPT 2016 - Volume 9665*, page 311–343, Berlin, Heidelberg, 2016. Springer-Verlag.

[MR07] Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on gaussian measures. *SIAM J. Comput.*, 37(1):267–302, apr 2007.

[MST90] Martello, Silvano, and Paolo Toth. *Knapsack problems: algorithms and computer implementations.* John Wiley & Sons, Inc., New York, NY, USA, 1990.

[MV22] Emmanouel T. Michailidis and Demosthenes Vouyioukas. A review on software-based and hardware-based authentication mechanisms for the internet of drones. *Drones*, 6(2), 2022.

[MW16] Daniele Micciancio and Michael Walter. Practical, predictable lattice basis reduction. In *Proceedings, Part I, of the 35th Annual International Conference on Advances in Cryptology — EUROCRYPT 2016 - Volume 9665*, page 820–849, Berlin, Heidelberg, 2016. Springer-Verlag.

[Ngu09] Phong Nguyen. *Hermite's Constant and Lattice Algorithms*, pages 19–69. 12 2009.

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

[NS97] Phong Q. Nguyen and Jacques Stern. Merkle-Hellman revisited: A cryptanalysis of the Qu-Vanstone cryptosystem based on group factorizations. In *Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings*, pages 198–212, 1997.

[NS98a] Phong Q. Nguyen and Jacques Stern. The Béguin-Quisquater server-aided RSA protocol from Crypto '95 is not secure. In *Advances in Cryptology - ASIACRYPT '98, International Conference on the Theory and Applications of Cryptology and Information Security, Beijing, China, October 18-22, 1998, Proceedings*, pages 372–379, 1998.

[NS98b] Phong Q. Nguyen and Jacques Stern. Cryptanalysis of the Ajtai-Dwork cryptosystem. In *Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings*, pages 223–242, 1998.

[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.

[NSS01] Phong Q. Nguyen, Igor E. Shparlinski, and Jacques Stern. Distribution of modular sums and the security of the server aided exponentiation. In Kwok-Yan Lam, Igor Shparlinski, Huaxiong Wang, and Chaoping Xing, editors, *Cryptography and Computational Number Theory*, pages 331–342, Basel, 2001. Birkhäuser Basel.

[NSS04] David Naccache, Nigel P. Smart, and Jacques Stern. Projective coordinates leak. In *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, pages 257–267, 2004.

[NV09] Phong Q. Nguyen and Brigitte Vallée. *The LLL Algorithm: Survey and Applications*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[NW21] Luca Notarnicola and Gabor Wiese. The hidden lattice problem. Cryptology ePrint Archive, Report 2021/1491, 2021. `https://ia.cr/2021/1491`.

[Odl90] Andrew M. Odlyzko. The rise and fall of knapsack cryptosystems. In *In Cryptology and Computational Number Theory*, pages 75–88. A.M.S, 1990.

[OY17] Muslum Ozgur Ozmen and Attila A. Yavuz. Low-cost standard public key cryptography services for wireless IoT systems. In *Proceedings of the 2017 Workshop on Internet of Things Security and Privacy*, page 65–70, New York, NY, USA, 2017. Association for Computing Machinery.

[OY18] Muslum Ozgur Ozmen and Attila A. Yavuz. Dronecrypt - an efficient cryptographic framework for small aerial drones. In *MILCOM 2018 - 2018 IEEE Military Communications Conference (MILCOM)*, pages 1–6, 2018.

[Pat95] Jacques Patarin. Cryptanalysis of the Matsumoto and Imai public key scheme of Eurocrypt'88. In *Annual international cryptology conference*, pages 248–261. Springer, 1995.

[Pat96] Jacques Patarin. Hidden fields equations (HFE) and isomorphisms of polynomials (IP): Two new families of asymmetric algorithms. In Ueli Maurer, editor, *Advances in Cryptology — EUROCRYPT '96*, pages 33–48, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[Pei09] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem: Extended abstract. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, STOC '09, page 333–342, New York, NY, USA, 2009. Association for Computing Machinery.

[Pei16] Chris Peikert. A decade of lattice cryptography. *Found. Trends Theor. Comput. Sci.*, 10(4):283–424, mar 2016.

[PU19] Robi Pedersen and Osmanbey Uzunkol. Secure delegation of isogeny computations and cryptographic applications. In *Proceedings of the 2019 ACM SIGSAC Conference on Cloud Computing Security Workshop*, CCSW'19, page 29–42, New York, NY, USA, 2019. Association for Computing Machinery.

[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.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6), sep 2009.

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

[Sch90] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *Advances in Cryptology — CRYPTO' 89 Proceedings*, pages 239–252, New York, NY, 1990. Springer New York.

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

[vDGHV10] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Monaco / French Riviera, May 30 - June 3, 2010. Proceedings*, pages 24–43, 2010.

[ZAR17] Kai Zhou, M. H. Afifi, and Jian Ren. Expsos: Secure and verifiable outsourcing of exponentiation operations for mobile cloud computing. *IEEE Transactions on Information Forensics and Security*, 12(11):2518–2531, 2017.

[ZJL+21] N. Zhang, Qi Jiang, Long Li, Xindi Ma, and Jianfeng Ma. An efficient three-factor remote user authentication protocol based on BPV-fourQ for internet of drones. *Peer-to-Peer Netw. Appl.*, 14:3319–3332, 2021.