Hypercubic Lattice Reduction and Analysis of GGH and NTRU Signatures

Michael Szydlo

RSA Laboratories, Bedford, MA, USA
mszydlo@rsasecurity.com

Keywords: Lattice Isomorphism, Lattice Distinguishing Oracle, Distributed Lattice Reduction, Decisional Lattice Problem, Gram Matrix Factorization, Integral Lattice Embedding Orthogonal Lattice, GGH Cryptanalysis, NTRUSign.

Abstract. In this paper, we introduce a new lattice reduction technique applicable to the narrow, but important class of *Hypercubic* lattices, ($L \cong \mathbb{Z}^N$). Hypercubic lattices arise during transcript analysis of certain GGH, and NTRUSign signature schemes. After a few thousand signatures, key recovery amounts to discovering a hidden unitary matrix U, from its Gram matrix $G = UU^T$. This case of the *Gram Matrix Factorization Problem* is equivalent to finding the shortest vectors in the hypercubic lattice, L_G, defined by the quadratic form G. Our main result is a polynomial-time reduction to a conjecturally easier problem: the *Lattice Distinguishing Problem.* Additionally, we propose a heuristic solution to this distinguishing problem with a distributed computation of many "relatively short" vectors.

1 Introduction

This paper discusses lattice reduction of *Hypercubic Lattices*, which are rotations of $\mathbb{Z}^N$ in Euclidean space. The Gram matrix of such a lattice is always of the form $G = UU^T$, for an integral unitary matrix U. An equivalent formulation of this problem is: given the Gram matrix $G = UU^T$, recover U, up to a signed permutation of its coordinates. To approach this problem, we introduce an a-priori easier problem: deciding whether two Gram matrices represent isomorphic lattices. We model the solution to this decisional problem with a *Lattice Distinguishing Oracle*, (LDO). Our principal result is a new oracle-algorithm to reduce hypercubic lattices, thus factoring $G = UU^T$, after making a polynomial number of calls to a Lattice Distinguishing Oracle.

Signature Schemes: The study of hypercubic lattices is motivated by cryptanalysis of two lattice based signature schemes: GGH, and (one version of) NTRUSign. The first, proposed by Goldreich, Goldwasser and Halevi [11] is based on the hardness of the closest vector problem (CVP) in a relatively general lattice. The second, by Howgrave-Graham, et. al. [12], is also based on the CVP, but it specifically chooses a class of lattices which have compactly describable bases. Both signature schemes are related to the corresponding, more well known, encryption schemes.

One measure of the security of lattice based schemes is the difficultly of reducing the underlying lattice involved. However, neither GGH nor NTRUSign have a security reduction to the underlying problem. As observed in [10], and reviewed below, a transcript of valid signatures of each (unmodified)[1] scheme necessarily leaks important information: the product of the private basis matrix with its transpose. This information can be used to shift the security to an apparently easier problem: the reduction of an auxiliary hypercubic lattice.

Combinatoric Approaches: The lattices we consider are not always presented with a basis of vectors with integer coordinates. This motivates a careful study of lattices presented with a Gram matrix. For such lattices, there is no Hermite Normal Form, and thus the question of whether or not two lattices are isomorphic is not easy. Both our reduction proof and discussion of the Lattice Distinguishing Problem involve novel, combinatoric approaches to special cases of more well known lattice reduction problems, when the lattices are defined by Gram matrices.

1.1 Our Contributions

We begin by discussing the equivalence of the hypercubic lattice reduction problem, a case of the Gram matrix factorization problem, and a case of the integral lattice embedding problem, and introduce the new *Lattice Distinguishing Problem*, which is of central importance in this paper. Secondly, we present a new algorithm which can solve the Gram matrix factorization problem by making a polynomial number of calls to a *Lattice Distinguishing Oracle*. This is the principal result of the paper. The number or oracle calls is bounded by kN^3, where k is the maximum bit length of the entries of the solution matrix U, and N is the dimension of this matrix. Thirdly, we show how to design a heuristic distinguishing oracle for the cases our algorithm requires. This construction requires a distribution of lattice vectors of length approximately $O(\sqrt{N})$ times the shortest vector.

This last construction is of additional interest for several reasons. First, it suggests a distributed approach to the Lattice Distinguishing Problem. Second, this $O(\sqrt{N})$ bound is of theoretical interest to complexity theory. It relates to the complexity results of Ajtai [1], and Micciancio[21] which suggest the SVP and approximate $\sqrt{2}$ - SVP lattice problems are in general difficult (NP-hard) problems. It also relates to the results of Goldreich and Goldwasser [6] which suggests that the $\sqrt{N}$ approximate vector problem is *unlikely* to be NP hard.

1.2 Organization

The rest of this paper is organized as follows. In Section 2, we recall some background on lattices and the associated computational problems. We also review the GGH-NTRUSign cryptanalysis. In Section 3, we formalize the definition of a

[1] There have been a variety of *perturbation* techniques proposed to reduce, or alter, the information leaked, see [12].

Lattice Distinguishing Oracle, and show how it can be used to solve some interesting problems. In Section 4, we show how this oracle provides a new strategy for the embedding problem. In Section 5, we present the principal result: a polynomial reduction of SVP in hypercubic lattices to the decisional lattice problem. In Section 6, we show how to heuristically design an LDO using distributions of lattice vectors. In Section 7, we conclude with a complexity theoretic interpretation, and we make comments on the ramifications for the security of the GGH and NTRUSign signature schemes. Finally, in the Appendix, we provide additional material on theta functions, which describe the vector length distributions of interest to the LDO design.

2 Background and Notation

In this section we present background on lattices and review how cryptanalysis of GGH and NTRUSign is related to the Gram matrix factorization problem.

2.1 Lattices

We begin with standard definitions and some notation used with Lattices. We define a general *lattice* to be discrete subgroup of Euclidean space, $\mathbb{R}^N$. A *basis* for a lattice is a set of vectors $\{\mathbf{v_i}\}$, $i \in \{1, \dots k \leq N\}$ such that each lattice point is a unique integer linear combination of the $\{\mathbf{v_i}\}$. The integer k is the *dimension* of the lattice, and usually $k = N$. These vectors may be described with the rows of a *basis matrix* B. Any other basis matrix B' is related to B by a *unitary transformation* $B' = UB$, where $U \in SL_N(\mathbb{Z})$ is called the *change of basis matrix.*

A basis B also defines a *positive definite symmetric quadratic form* given by its *Gram matrix*, $G = BB^T$, where the matrix $G = (g_{i,j})$, specifies an *inner product* $\mathbf{v_i} \cdot \mathbf{v_j} = g_{i,j}$. Conversely, each positive definite symmetric matrix, G, defines an abstract lattice, L_G, as the span of a basis $\{\mathbf{v_i}\}$ satisfying the inner product specified by G. The *Gram Schmidt orthogonalization process* efficiently computes an *embedding* $\sigma : L_G \to \mathbb{R}^N$, realizing $\sigma(L_G)$ concretely in $\mathbb{R}^N$. Such an embedding is determined by L up to an element $\phi \in O_N(\mathbb{R})$, the *orthogonal group.* Two lattices are *isomorphic*, $(L_1 \cong L_2)$, if there is a distance preserving map $\phi : L_1 \to L_2$ between them. If both lattices are contained in the same space $\mathbb{R}^N$, such an isomorphism is a *rotation*, given by an element $\phi \in O_N(\mathbb{R})$.

A lattice is called an *integral lattice* if it is isomorphic to L_G for some integral Gram matrix $G \in M_n(\mathbb{Z}^N)$. If a lattice L is N-dimensional and has an *integral embedding*: $\sigma : L \to \mathbb{Z}^N$, it is called an *integer coordinate lattice.* Such a lattice determines the integral embedding up to a *signed permutation*, $\phi \in O_N(\mathbb{Z})$. Integral lattices are convenient for computation, and some applications only consider lattices which are subsets of $\mathbb{Z}^N$, the *trivial lattice.* We define a *hypercubic lattice* to be a lattice $L \cong \mathbb{Z}^N$, i.e., as a subset of $\mathbb{R}^N$, it is a rotation of the trivial lattice.

2.2 Lattice Problems

We now review some computational lattice problems of interest to cryptography.

Standard Reduction: The problem of *Lattice Reduction* is the problem of replacing one basis with a better one [20], whose vector are shorter and more orthogonal. A reduced basis helps to solve the *shortest vector problem* (SVP), which seeks a shortest vector in the lattice, and the *closest vector problem* (CVP), which seeks a lattice vector closest to a point $p \in L \otimes \mathbb{Q}$ not in the lattice. The fundamental tools used to solve these and other integer lattice problems are the LLL reduction algorithm, and its variants [20].

Reducing Gram Matrices: Many implementations of lattice reduction require an integral basis B as input and output a transformation matrix U, and the more reduced basis B', which is equal to UB. However, the reduction algorithms such as LLL do not require an integral embedding as input; they operate equally well on lattices L_G, defined a the Gram matrix G. These general reduction algorithms take as input a basis specified by an integral Gram matrix G, and produce a transformation matrix U, and the Gram matrix G' of the more reduced basis, so that $G' = UGU^T$. For example, the implementation NTL requires an integral basis, while Pari accepts any integral Gram matrix as input.

Integral Embeddings: There are other interesting computational problems for lattices L_G defined by Gram matrices G. The *Lattice embedding problem* seeks an embedding $\sigma : L_G \to \mathbb{Z}^N$. This is equivalent to finding an integral basis B such that $G = BB^T$, so this problem is also called the *Gram matrix factorization problem.* A decisional problem of central interest for this paper is the *Lattice isomorphism problem*: Given two lattices L_G, and $L_{G'}$, defined by Gram matrices G, and G', determine if $L_G \cong L_{G'}$. This is equivalent to determining whether or not there exists a transformation matrix U such that $G' = UGU^T$, so another appropriate name for this problem is the *Decisional lattice conjugacy problem.* Note that the lattice isomorphism problem is much easier when given integral bases: the lattices are isomorphic if and only if they have the same Hermite Normal Form (HNF).

Hypercubic Lattice Case: This paper focuses on hypercubic lattices, L_G, defined by a Gram matrix G. By definition, a hypercubic lattice is isomorphic to $\mathbb{Z}^N$, so it must have a Gram matrix of the form $G = UU^T$, where U is a unitary integer matrix. We review three formulations of this problem.

Proposition 1. Hypercubic Lattice Equivalence
Let $G = UU^T$ be the Gram matrix of an integral unitary basis matrix U, and let L_G be the associated hypercubic lattice. The following computational problems are equivalent:
A. *Given G, find the shortest vectors in L_G.*
B. *Given G, recover U, up to sign and order of the coordinates.*
C. *Given G, construct an embedding $L_G \to \mathbb{Z}^N$.*

We present the simple proof in Appendix A, and note that this equivalence applies only to these very specific lattices. In general, when considering computational problems we keep in mind that the difficulty depends on the instance distribution, and that special cases often turn out to be easier.

2.3 GGH and NTRUSign Transcript Application

Here we summarize the observation made in [10] that a transcript of NSS or NTRUSign essentially recovers a Gram matrix of a certain lattice. In some sense, the GGH and NTRU Signature schemes are adaptations of the analogous encryption schemes. However, it has been more difficult to link the security of the signature schemes to the underlying computational lattice problem.

We now review just enough details of the schemes to recall how the transcript averaging attack works. GGH and NTRUSign are based on the difficulty of a closest vector problem in a certain lattice. The main difference between GGH and NTRUSign is that lattices in NTRUSign are chosen from a more restrictive class ("bi-circulant") in order to achieve more efficient computation and storage. Each scheme employs a lattice with a public basis matrix B, and a private basis matrix M, related by $B = UM$, where $det(U) = 1$. The signing process involves mapping a hashed message to a vector $\mathbf{m} \epsilon \mathbb{Z}^N$, randomly according to some distribution. The private basis M is used to compute a vector very close to m. This is essentially accomplished by rounding $\mathbf{m}M^{-1}$ to an integer valued vector $\mathbf{w}$, and setting $\mathbf{s} = \mathbf{w}M$. Since the private basis M is nearly orthogonal, $\mathbf{s}$ is close it $\mathbf{m}$, and the verifier checks that the norm $|\mathbf{m} - \mathbf{s}|$ is below some threshold. See [11] and [12] for further details of this process.

One by-product of the projection of random message representatives $\mathbf{m}$ is that, to a sufficient approximation, the coefficients of $\mathbf{w}$ are symmetrically distributed, and nearly independent. That is, for two different coordinates w_i, and w_j, the average dot product $w_i \cdot w_j$ is zero. On the other hand, the average squared lengths $w_i \cdot w_i$ are all equal to some positive constant (say K). Armed with such transcript, the analyst computes the $N \times N$ matrix $s^T s$, for each signature $\mathbf{s}$, (considered as a row vector), and averages them.

$$Avg(\mathbf{s}^T\mathbf{s}) = Avg(M^T\mathbf{w}^T\mathbf{w}M) = M^T Avg(\mathbf{w}^T\mathbf{w})M. \tag{1}$$

To a sufficient approximation, the diagonal entries of $\mathbf{w}^T\mathbf{w}$ converge to K, and the others converge to 0, so the average of $\mathbf{w}^T\mathbf{w}$ is about K times the identity matrix. The matrix average is KM^TM, a multiple of M^TM, which is the Gram matrix of the transpose of M. Thus the adversary obtains the *Gram Matrix of the Private Transpose Basis.* Combining this with the public basis B, we can compute

$$G = B(M^TM)^{-1}B^T = UMM^{-1}M^{-T}M^TU^T = UU^T. \tag{2}$$

If we now let U play the role of a basis matrix, we see that the cryptanalyst has G, the *Gram Matrix of the Transformation Basis* U. Because U is the transformation matrix between public and private bases, ($B = UM$), key recovery amounts to factoring $G = UU^T$,(up to sign and permutation of the coordinates). The cryptanalyst can also consider the equivalent problems of finding the shortest vectors in L_G, or finding an integral embedding $\sigma : L \rightarrow \mathbb{Z}^N$.

3 LDO and Parity Testing

We have already introduced the important lattice isomorphism problem above, and in this section, we formally define an oracle to solve it. We will see that this oracle can be used to solve a number of interesting problems. Our oracle considers two lattices, L_G, and $L_{G'}$, defined by Gram matrices G, and G', and determines if they are isomorphic.

Definition 1. *A* **Lattice Distinguishing Oracle (LDO)**
is an efficient algorithm which computes the following function:
Input: G_1, G_2.
Output: *True if* $L_{G_1} \cong L_{G_1}$*, or* $False$ *otherwise.*

Notation. We first collect some notation to work with the Gram matrix G, and potential integral embeddings $\sigma : L_G \rightarrow \mathbb{Z}^N$. Recall that L_G is defined as the span of N abstract vectors, denoted $\{\mathbf{v_i}\}$, whose dot products $\mathbf{v_i} \cdot \mathbf{v_j}$ are defined by the entries of G. Note that σ is only determined by G up to a signed permutation of the coordinates[2]. For a particlar embedding, we denote the images $\sigma(\mathbf{v_i})$ by $\hat{\mathbf{v_i}}$. Since we have no integral coordinates for the $\{\mathbf{v_i}\}$, we simply record linear combinations $\mathbf{w} = \Sigma a_i \mathbf{v_i}$, $(a_i \in \mathbb{Z})$ with the vector $\mathbf{a}$ having coefficients $\{a_i\}$. Let T be a matrix whose rows consist of such coefficient vectors. If the corresponding vectors are independent, T describes a lattice *in terms of the* $\{\mathbf{v_i}\}$. Then the Gram matrix of this *auxiliary lattice* is $G(T) = TGT^T$.
Information from G Keeping in mind our interest in factoring $G = UU^T$, we expect to be able to learn some properties of the rows of U which are invariant under signed permutation. For example, we know $\mathbf{v_i} \cdot \mathbf{v_i}$, the squared lengths of of these vectors, from the diagonal of G. We are most interested in certain well defined "parity" measures of a vector $\mathbf{w} = \Sigma a_i \mathbf{v_i}$, $(a_i \in \mathbb{Z})$, which we define as follows:

Parity Measures

Let $\lambda_1(\mathbf{w})$ be the number of coordinates of $\hat{\mathbf{w}}$ congruent to 1 (mod 2).
Let $\lambda_2(\mathbf{w})$ be the number of coordinates of $\hat{\mathbf{w}}$ congruent to 2 (mod 4).
Let $\lambda_k(\mathbf{w})$ be the number of coordinates of $\hat{\mathbf{w}}$ congruent to 2^{k-1} (mod 2^k).

Computing λ_1 from the LDO We would like to know how many of the coordinates of $\hat{\mathbf{v_1}}$ are odd. We remark that we immediately know this number (mod 4) from its squared length. To learn the actual number we consider the following auxiliary lattice: the span of the vectors $\{\mathbf{v_1}, \mathbf{2v_2}, \ldots \mathbf{2v_N}\}$. We have no coordinates for this lattice, but we know its Gram matrix $G_{aux} = AGA^T$, where A is the diagonal matrix with $A_{1,1} = 1$, and $A_{i,i} = 2$ for $i \geq 2$. We also know that under any integral embedding σ, one basis of $2L_G$ is twice the identity matrix, $2Id$. It is also easy to see that that under any embedding σ, the basis defined by G_{aux} has a Hermite Normal Form with a very special form: the first row has only entries in $\{0, 1\}$, and all other nonzero entries are 2's on the diagonal. Up

[2] So the σ's may be also considered as coordinate permutations of the basis U.

to isomorphism, there are only N possibilities for the lattice defined by G_{aux}, depending on the number of 1's. Next we simply "artificially" create a Gram matrices for each such lattice, and denote them $Test_i$ ($1 \leq i \leq N$). With these auxiliary lattices, we can determine $\lambda_1(v_1)$ with the following oracle algorithm:

Computing $\lambda_1(G_{aux})$ with the LDO

```
For i=1 to N
If $LDO(G_{aux}, Test_i) = True$,   output $i$.
Loop
Otherwise, output ERROR
```

Thus, we can obtain $\lambda_1(\mathbf{v_1})$ with N oracle calls. The same approach will compute $\lambda_1(\mathbf{w})$ for any vector $\mathbf{w}$ which is a linear combination of the $\{\mathbf{v_i}\}$. We remark that by using the knowledge of the squared length (mod 4) the algorithm may be sped up by a factor of 4.

Secondary Tests: The computation of the other λ_k quantities can be similarly performed. There is one new detail which appears: In order to limit the number of isomorphism classes to N, we compute $\lambda_2(\mathbf{w})$ only for vectors $\mathbf{w}$ for which $\hat{\mathbf{w}}$ has only a single coordinate congruent to 1 (mod 4). This way, there are still only N test lattices needed. and the Gram matrices $Test_i$ are created from the N possible Hermite normal forms of the lattices spanned by the (linearly dependent) $\{\mathbf{w}, \mathbf{4v_1}, \mathbf{4v_2}, \ldots \mathbf{4v_N}\}$. As above, the Gram matrix G_{aux} is computed by conjugating G with the appropriate transformation matrix. Such a transformation matrix is easy to find, but it does involves removing a linear dependency.

In this manner, successive quantities $\lambda_k(\mathbf{w})$ can also be evaluated for $k > 2$. In these cases attention is limited to vectors $\mathbf{w}$ for which $\hat{\mathbf{w}}$ has one odd coordinate, and the rest congruent to 2^{k-1} (mod 2^k). Then, as above there will be N candidate lattices to compare with the lattice spanned by G_{aux}.

4 The Embedding Strategy

In this section we describe a strategy to embed the vectors $\mathbf{v_i}$ into $\mathbb{Z}^N$ by using the parity testing that the LDO provides us with. Henceforth, we allow ourselves to compute $\lambda_k(\mathbf{w})$ for certain vectors $\mathbf{w}$ which are linear combinations of the $\mathbf{v_i}$. To give the reader an intuitive feel for the process, we begin very explicitly, constructing consistent embeddings of $\mathbf{v_1}$, $\mathbf{v_2}$, and $\mathbf{v_3}$. We also illustrate these early steps with a numerical example.

We begin the construction of an embedding $\sigma : L_G \to \mathbb{Z}^N$ with $\mathbf{v_1}$, using knowledge of $\lambda_1(\mathbf{v_1})$. We can write down $\hat{\mathbf{v_1}}$ (mod 2) for some σ by simply letting $\hat{\mathbf{v_1}}$ (mod 2) be the vector whose first $\lambda_1(\mathbf{v_1})$ coordinates are 1, and the rest zero. These values must be correct for some integral embedding σ, but not for every such embedding. By choosing the first coordinates of $\hat{\mathbf{v_1}}$ to be equal to 1 (mod 2), we have effectively partially chosen σ. We illustrate this with a toy example where we take $N = 10$, and assume $\lambda_1(\mathbf{v_1}) = 6$. We then write

$$\hat{\mathbf{v_1}} = (1, 1, 1, 1, 1, 1, 0, 0, 0, 0) \quad (\mathit{mod}\ 2).$$

Next we would like to continue, and choose $\hat{\mathbf{v_2}}$ (mod 2), based on $\lambda_1(\mathbf{v_2})$. However, we want to do this *consistently* with our choice of $\hat{\mathbf{v_1}}$ (mod 2). In other words, there should exist a single σ which maps $\mathbf{v_1}$ to $\hat{\mathbf{v_1}}$, and also $\mathbf{v_2}$ to $\hat{\mathbf{v_2}}$. The missing information we need to do this is the number of coordinates for which *both* $\hat{\mathbf{v_1}}$ and $\hat{\mathbf{v_2}}$ are odd. We can obtain this additional information from $\lambda_1(\mathbf{v_1} + \mathbf{v_2})$, which is clearly $\lambda_1(\mathbf{v_1}) + \lambda_1(\mathbf{v_2})$ minus twice the number of overlapping odd coordinates.

We continue our example, supposing additionally that $\lambda_1(\mathbf{v_2}) = 5$, and $\lambda_1(\mathbf{v_1} + \mathbf{v_2}) = 3$. We calculate twice the number of overlapping odd coordinates to be $5 + 6 - 3 = 8$, so there are four. We can then consistently write

$$\hat{\mathbf{v_2}} = (1, 1, 1, 1, 0, 0, 1, 0, 0, 0) \quad (\mathit{mod}\ 2).$$

Continuing with $\mathbf{v_3}$, we see again that $\lambda_1(\mathbf{v_3})$ is not enough information to write a consistent $\hat{\mathbf{v_3}}$. We view the previous choices as effectively dividing the N coordinates into four "regions", namely the four possibilities for $(\hat{\mathbf{v_1}}, \hat{\mathbf{v_2}})$, which are $(1, 1)$, $(1, 0)$, $(0, 1)$, and $(0, 0)$. We want the number of odd coordinates of $\hat{\mathbf{v_3}}$ in each region, so we define four variables X_1, X_2, X_3, and X_4 to represent these quantities. Since $\lambda_1(\mathbf{v_3})$ determines the total number of 1's, it provides one constraint, namely $X_1+X_2+X_3+X_4$. The quantities $\lambda_1(\mathbf{v_3} + \mathbf{v_1})$, $\lambda_1(\mathbf{v_3} + \mathbf{v_2})$, and $\lambda_1(\mathbf{v_3} + \mathbf{v_2} + \mathbf{v_1})$ provide three more constraints. These equations are linearly independent, so the solutions may be found with Gaussian elimination.

Continuing our example, suppose $\lambda_1(\mathbf{v_3}) = 6$, $\lambda_1(\mathbf{v_3} + \mathbf{v_1}) = 6$, $\lambda_1(\mathbf{v_3} + \mathbf{v_2}) = 7$, and $\lambda_1(\mathbf{v_3} + \mathbf{v_2} + \mathbf{v_1}) = 7$. One can check that $X_1 = 2$, $X_2 = 1$, $X_3 = 0$, and $X_4 = 3$ are the solutions for this example. Thus we can write

$$\hat{\mathbf{v_3}} = (1, 1, 0, 0, 1, 0, 0, 1, 1, 1) \quad (\mathit{mod}\ 2).$$

The number of region variables may expand up to a maximum N, (one for each coordinate position) when dealing with the subsequent vectors $\mathbf{v_i}$, $(i > 3)$. Of course, the number of independent constraints collected must equal the number of variables in order to solve the system. As before, these constraints come from $\lambda_1(\mathbf{v_i})$, and enough other values of $\lambda_1(\mathbf{v_i} + \mathbf{v'})$ where $\mathbf{v'}$ ranges over nonempty sums of the $\mathbf{v_j}$, $(j < i)$. Note that one selects appropriate vectors $\mathbf{v'}$ which guarantee linear independence before calling the oracle.

The above procedure and example are representative of our strategy to embed the vectors $\mathbf{v_i}$. We essentially use the LDO to obtain selected λ_k values which allow us to write down approximate values of $\{\hat{\mathbf{v_i}}\}$ which are consistent with an embedding σ. The procedure explained in this section is a major component of the full embedding algorithm, which we define in the next section. In fact we have provided an oracle algorithm which given G, produces a set of vectors $\{\hat{\mathbf{v_i}}\}$ (mod 2), which are the images $\sigma(\mathbf{v_i})$, for some embedding σ.

Let us keep in mind the maximum number or oracle calls this step has required. For each of the N vectors $\mathbf{v_i}$, up to N values λ_1 were required, and each of these required up to N LDO calls. In total, this step has cost at most than N^3 LDO calls.

5 Hypercubic Embedding Algorithm

In this section we describe further techniques which, when combined with the step described in the previous section, yield a complete embedding $\sigma : L_G \rightarrow \mathbb{Z}^N$. This will complete our main result, that the equivalent problems of Hypercubic lattice reduction, Gram matrix factorization problem, and Embedding problem are polynomial time reducible to the lattice distinguishing problem. Choosing one formulation, we prove

Theorem 1. *Let U be an N-dimensional unitary matrix, and let k be the maximum bit-length of the entries, and let $G = UU^T$. Then, given G, U may be recovered, up to sign and order of the coordinates, by making at most than kN^3 calls to a Lattice Distinguishing Oracle.*

Notation. We continue to use G, U, $\mathbf{v_i}$, $\hat{\mathbf{v}}_\mathbf{i}$, and λ_k as defined above, and as before, $\sigma : L_G \rightarrow \mathbb{Z}^N$, will always be an integral embedding. Additionally, for each positive integer k, let $T^{(k)}$ be an integer $N \times N$ matrix, with elements denoted $\{t^{(k)}_{i,j}\}$. Let $\mathbf{w_i}^{(k)} = \varSigma\, t^{(k)}_{i,j}\, \mathbf{v_i}$, and when an embedding σ is implied, we let $\hat{\mathbf{w}}_\mathbf{i}^{(k)} = \sigma(\mathbf{w_i}^{(k)})$. Finally, let k_0 be the maximum bit length of the entries in U. With this notation in place we can outline the major steps of our algorithm.

Algorithm 2 *Embedding Algorithm*

```
1. Find $\{\hat{\mathbf{v}}_\mathbf{i}\} \in \mathbb{Z}^N \mid \exists\sigma$ with $\hat{\mathbf{v}}_\mathbf{i} = \sigma(\mathbf{v_i})$ (mod 2).
2. Find $T^{(1)}$ defining $\mathbf{w_i}^{(1)} \mid \exists\sigma$ with $\hat{\mathbf{w}}_\mathbf{i}^{(1)} = \mathbf{e_i}$ (mod 2).
3. Find $T^{(2)}$ defining $\mathbf{w_i}^{(2)} \mid \exists\sigma$ with $\hat{\mathbf{w}}_\mathbf{i}^{(2)} = \mathbf{e_i}$ (mod 4).
4. For each $3 \leq k \leq k_0$ Do,
   find $T^{(k)}$ defining $\mathbf{w_i}^{(k)} \mid \exists\sigma$ with $\hat{\mathbf{w}}_\mathbf{i}^{(k)} = \mathbf{e_i}$ (mod $2^k$).
5. For each pair of indices $i, j$, compute the dot products $d_{i,j} = \mathbf{v_j} \cdot \mathbf{w_i}^{(k_0)}$.
6. Output, $\{\hat{\mathbf{v}}_\mathbf{i}\}$ where the j'th coordinate of $\hat{\mathbf{v}}_\mathbf{i}$ is $d_{i,j}$, reduced to the smallest representative (mod $2^k$).
```

Step 1. This step has been discussed in detail in Section 4, where the values λ_1 were used to find vectors $\{\hat{\mathbf{v}}_\mathbf{i}\}$. We remark that this step is the most interesting, due to the linear algebra not required in subsequent steps. We also note that among the finite number of possible embeddings, σ, the choices made in this step have fixed the order of the coordinates. Only the sign ambiguity remains.

Step 2. Consider the matrix V defined by the rows of $\{\hat{\mathbf{v}}_\mathbf{i}\}$. As V's is congruent to $\sigma(\mathbf{v_i})$ (mod 2), and the latter has determinant one, V is invertible when considered over the field of two elements. We let $T^{(1)}$ be the integer matrix with entries 0 and 1, which represents this inverse. Now $T^{(1)}$ defines the vectors

$\mathbf{w_i}^{(1)} = \Sigma\, t_{i,j}^{(1)}\, \mathbf{v_i}$. Thus the images of the $\mathbf{w_i}^{(1)}$ under σ are the rows of the matrix $T^{(1)}V$, which is the identity, (mod 2). So we conclude that $\hat{\mathbf{w}}_\mathbf{i}^{(1)} = \mathbf{e_i}$ (mod 2). These vectors $\mathbf{w_i}^{(1)}$ will be used in subsequent steps.

Step 3. In this step we will improve the vectors $\mathbf{w_i}^{(1)}$, producing vectors $\mathbf{w_i}^{(2)}$, so that $\hat{\mathbf{w}}_\mathbf{i}^{(2)} = \mathbf{e_i}$ (mod 4). For a fixed σ satisfying the conditions of Step 1, and our choice of $\{\mathbf{w_i}^{(1)}\}$, the i'th coordinate of $\sigma(\mathbf{w_i}^{(1)})$ is odd, thus ± 1 (mod 4). However, as of step 1, σ had been only been determined up to sign. Among the 2^N choices, there is always one for which the i'th coordinate of $\sigma(\mathbf{w_i}^{(2)})$ is 1. We retain this choice of σ throughout the remainder of this algorithm. Therefore, our choices have also fixed $\hat{\mathbf{v}}_\mathbf{i} = \sigma(\mathbf{v_i})$.

For each index i, we initially set $\mathbf{w_i}^{(2)} = \mathbf{w_i}^{(1)}$ and proceed to modify it. Recalling Section 3, we learned that the LDO may be used to compute $\lambda_2(\mathbf{w_i}^{(2)})$, since $\hat{\mathbf{w}}_\mathbf{i}^{(2)}$ meets the stated criteria: One coordinate is equal to 1 (mod 4), and the rest are equal to 2 or 0 (mod 4). The value $\lambda_2(\mathbf{w_i}^{(2)})$ lets us measure the number of two's in $\hat{\mathbf{w}}_\mathbf{i}^{(2)}$. Now, for each index $j \neq i$, we tentatively add $\mathbf{2w_j}^{(1)}$ to it. Notice that $\hat{\mathbf{2w_j}}^{(1)} = 2$ (mod 4) only in one spot, at the j'th coordinate, so the operation of adding $\mathbf{2w_j}^{(1)}$ always modifies $\lambda_2(\mathbf{w_i}^{(2)})$ by 1. If it decreases we keep it, otherwise not, and so after all $j \neq i$ are considered, $\lambda_2(\mathbf{w_i}^{(2)})$ is reduced to zero. These vector additions are recorded in terms of integral combinations of the $\mathbf{v_i}$, so eventually we obtain the matrix $T^{(2)}$ defining the final $\mathbf{w_i}^{(2)}$. Now that $\hat{\mathbf{w}}_\mathbf{i}^{(2)} = \mathbf{e_i}$ (mod 4), and we proceed to the next step. Notice that Step 3 also completes with at most than N^3 LDO calls.

Step 4. This step consists of a procedure which is repeated for each $k \geq 3$ up to the bound k_0. As with Step 3, the goal is to further improve each of the $\mathbf{w_i}$. Unlike Step 3, however, σ has already been fixed, so when we begin by setting $\mathbf{w_i}^{(k)} = \mathbf{w_i}^{(k-1)}$, the the i'th coordinate of $\sigma(\mathbf{w_i}^{(k)})$ can not be automatically assumed to be congruent to 1 (mod 2^k). Instead, it might be $1 + 2^{k-1}$ (mod 2^k), and this can be easily tested, since then the squared length of $\mathbf{w_i}^{(k)}$ will not be 1 (mod 2^{k+1}). If this happens, the first step is to multiply $\mathbf{w_i}^{(k)}$ by the scalar $2^{k-1} + 1$.

Once in this form, we can evaluate $\lambda_3(\mathbf{w_i})$. We take the same strategy as above, and add certain vectors $\mathbf{2^{k-1}w_j}^{(1)}$ to it, until $\lambda_3(\mathbf{w_i}^{(k)})$ is reduced to zero, at which point we have our $T^{(k)}$ defining the final $\mathbf{w_i}^{(k)}$.

Step 5. In this step we compute the dot products $d_{i,j} = \mathbf{v_j} \cdot \mathbf{w_i}^{(k_0)}$. To do this, we write each $\mathbf{w_i}^{(k_0)}$ as a linear combination of the $\{\mathbf{v_j}\}$ according to the rows of $T^{(k)}$, and use the fact that we know each $\mathbf{v_i} \cdot \mathbf{v_j}$ from the Gram matrix G. More simply put, we perform a matrix multiplication $T^{(k)}G$.

Step 6. We now use the fact that σ is an embedding, to reason that $\mathbf{v_j} \cdot \mathbf{w_i}^{(k_0)}$, which we know, also equals $\hat{\mathbf{v}}_\mathbf{j} \cdot \hat{\mathbf{w}}_\mathbf{i}^{(k_0)}$, which equals the i'th coordinate of $\hat{\mathbf{v}}_\mathbf{j}$ (mod 2^{k_0}), since $\hat{\mathbf{w}}_\mathbf{i}^{(k_0)} = \mathbf{e_i}$ (mod 2^{k_0}). But by the assumption on the signed bit length of U, we actually know $\hat{\mathbf{v}}_\mathbf{j}$ over $\mathbb{Z}$. We have completed the computation of $\sigma : L_G \rightarrow \mathbb{Z}^N$, and the algorithm terminates after outputting the exact values $\hat{\mathbf{v}}_\mathbf{j}$.

This completes the algorithm and the proof of Theorem 1. All told, at most kN^3 calls to the lattice distinguishing oracle have been made. The hypercubic lattice reduction problem is polynomial time reducible to the lattice distinguishing problem.

6 Heuristic LDO Implementation

While the focus of this paper is the reduction proof, Theorem 1, it is natural to ask if such a lattice distinguishing oracle is feasible to implement. In this section, we discuss a heuristic approach to solving this decision problem for the specific lattices required in our reduction. We remark immediately, that even if the LDO is practical, the potentially very large number, (kN^3), of oracle calls might still leave the hypercubic lattice reduction a difficult problem. This should not be too disappointing, given the exponential nature of the general algorithms to compute *exact* shortest lattice vectors. On the other hand, if only Steps 1 and 2 of the algorithm 2 are completed, the difficulty of the original shortest vector problem can be made significantly easier.

So how hard is it to implement an LDO? Our heuristic algorithm to realize this oracle will not treat the general problem, but instead will focus on the cases required for our algorithm. In fact, our oracle will only need to distinguish between N lattices at a time. Recalling the discussion in section 3, we used the LDO to compare an unknown lattice defined by the Gram matrix G_{aux}, with N potential lattices, $Test_i$, $(1 \leq i \leq N)$. This auxiliary lattice defined by G_{aux} depends on some input vector defined in terms of the $\{\mathbf{v_i}\}$. On the other hand, the N matrices $\{Test_i\}$ were specifically constructed from a known integer basis. By construction the lattice of G_{aux}, must be the lattice of one $Test_i$. Because we know the structure of the candidates, $Test_i$, we directly see in what ways they differ.

6.1 Modular Tests

Our approach relies on the fact that the N candidate lattices enjoy a different distribution of vectors. For the simplest possible example, it is clear that among the N possibilities, only $Test_1$ has a vector of length 1. However, it is not easy to check whether the lattice of G_{aux} has such a vector. Finding it, at least, requires the solution of the SVP itself! Fortunately, other distinguishing features of these distributions can be perceived with larger vectors. The most prominent example of this considers the lengths of the vector (mod 4). For example, let $\mathbf{z}$ be any vector in the lattice of G_{aux} which is not in $2L_G$. If the length of $\mathbf{z}$ squared is 1 (mod 4), then the lattice must be isomorphic with that of $Test_i$ for some $i = 1$ (mod 4). The same type of conclusion may be reached if the length squared is 0, 2 or 3 (mod 4). This discussion shows how the isomorphism question may be easy in certain cases, but in general, these tricks will not be sufficient.

6.2 Statistical Tests

We now explain how to exploit the difference of the distributions in general. By fixing some bound B, we consider only vectors of length squared less than this bound. For now, let us assume that it is possible to collect a large sample of vectors from the lattice defined by G_{aux} drawn nearly uniformly among all vectors in this lattice of squared length less than B. For each of the N possible test lattices, we also compute a similar distribution. We then compare the frequency distribution of the lengths from our sample drawn from the lattice defined by G_{aux} to the distributions corresponding to one of the $Test_i$ lattices. If the frequency distribution between two lattices matches closely enough, the lattices will be declared isomorphic.

6.3 Collecting vectors

The remaining question is how the sample distribution is collected. We want to use LLL or another lattice reduction algorithm to obtain a medium length vector. In practice, this is feasible for certain vector length bounds. We set our bound B, and reduce the lattice until a vector shorter than B is found.

This sample distribution of such vector lengths must be created as uniformly as possible. To encourage this, the basis is randomized before each lattice reduction begins. It is possible that other measures may also be needed. We conjecture that this procedure will produce a sample of sufficient accuracy to decide if two lattices are isomorphic.

6.4 Calculating Exact Distributions

In Appendix B we show how we can compute the *exact* distribution of vector lengths for the test lattices $Test_i$. This slightly simplifies the oracle construction, and additionally gives us some information about when a (uniformly distributed) sample size consisting of certain size vectors is likely to be sufficient for the oracle's decision.

6.5 Experiments

We performed experiments using the techniques in Appendix B to compute theta functions, which quantify the extent to which the distributions differ.

We tested lattices of dimension 100 and 500 for lattices defined by $Test_i$ for varying i values. We found that when B was reduced below about $3/10N$ times the squared length of the shortest vector, the differences in these distributions became perceptively different, so that a sample size of several thousand vectors would suffice to distinguish lattices of the form $Test_i$ from one another.

Our experiments suggested that a practical bound for the length of the sample vectors is some $O(\sqrt{N})$ multiple of the length of the shortest vector. This bound is merely conjectural, but is intriguing due to the relationship with the complexity results described in the introduction.

7 Conclusions

Our reduction of the hypercubic SVP to certain Lattice Distinguishing Problems is an interesting connection between a computational and a decisional problem. A large sequence of correct solutions to the decisional problem combines to solve the computational problem.

7.1 Security Ramifications

This work on reduction of hypercubic lattices shows that the transcript attacks reviewed above are indeed relevant to the security of the schemes. The combination of transcript averaging, and this algorithm, and the potential of a feasible Distinguishing Oracle, suggest a weakness in signature schemes which leak this specific kind of information. Currently, the *practical* security threat to the schemes is not clear - given the large number of Oracle calls and (albeit easier) lattice problems behind the LDO. Given the new approaches to the Gram matrix problem, it seems prudent to always use the perturbation methods with these schemes. This additional perturbation step may unfortunately reduce efficiency.

7.2 Complexity Ramifications

We hope that the special techniques for SVP hypercubic lattices may have analogues for general lattices. LDO may be realized given a collection of vectors of size about $\sqrt{\mathbf{N}}$ times the shortest vector - the same threshold that the work of Goldwasser suggests may be feasible. Combining these ideas, there is reasonable evidence to believe that SVP in hypercubic lattices is strictly easier than SVP in general lattices. Finally, the Lattice Distinguishing Problem may also be of independent interest and have other applications.

7.3 Further Work

The most relevant open question is still the feasibility of the LDO. For example, we would like to know if the sampling techniques yield sufficiently accurate distributions to realize an LDO.

We suggest a strategy to deal with an LDO which is allowed some chance of failure. Such a failure may be caught with high probability by repeatedly applying tests of consistency. One such check is $\lambda(\mathbf{v_1} + \mathbf{v_2}) \leq \lambda(\mathbf{v_1}) + \lambda(\mathbf{v_2})$. Proving that such a technique always works would be useful as it would increase the robustness of algorithms using an LDO.

The collection of sample vectors in our method of implementing an LDO may be completely distributed. Of course, an important problem would be the introduction of distributed algorithms for general lattices.

As a general research program, it should also be fruitful to study the difficulty of problems involving lattices with special structure. The hypercubic lattice problem is also a natural special case to consider, given its several equivalent formulations.

8 Acknowledgments

I would like to thank Craig Gentry, Phong Nguyen, and Jacques Stern for helpful and interesting discussions. I would also like to thank the anonymous reviewers for useful advice on how to clarify the presentation of these results.

References

1. M. Ajtai, *The shortest vector problem in L_2 is NP-hard for randomized reductions*, in Proc. 30th ACM Symposium on Theory of Computing, 1998, 10–19.
2. H. Cohen, A Course in Computational Algebraic Number Theory, Graduate Texts in Mathematics, 138. Springer, 1993.
3. D. Coppersmith and A. Shamir, *Lattice Attacks on NTRU*, in Proc. of Eurocrypt '97, LNCS 1233, pages 52–61. Springer-Verlag, 1997.
4. I. Dinur, G. Kindler, S. Safra, *Approximating CVP to within almost-polynomial factors is NP-hard*, in Proc. 39th Symposium on Foundations of Computer Science, pages 99–109, 1998.
5. N. Elkies, *Lattices, Linear Codes, and Invariants*, in Notices of the American Math. Society, 47 pages 1238–1245, Cambridge University Press, 2000.
6. O. Goldreich and S. Goldwasser, *On the Limits of Non-Approximability of Lattice*, In Proc. of the 13th ACM Symposium on the Theory of Computing, 1998.
7. O. Goldreich, D. Micciancio, S. Safra, J.P. Seifert, *Using Lattice Problem in Cryptography,*1999.
8. C. Gentry, J. Jonsson, J. Stern, M. Szydlo, *Cryptanalysis of the NTRU signature scheme*, in Proc. of Asiacrypt '01, LNCS 2248, pages 1–20. Springer-Verlag, 2001.
9. O. Goldreich, D. Micciancio, S. Safra, J.P. Seifert, *Approximating shortest lattice vectors is not harder than approximating closest lattice vectors*, Electronic Colloquium on Computational Complexity, 1999.
10. C. Gentry, M. Szydlo, *Cryptanalysis of the Revised NTRU signature scheme*, in Proc. of Eurocrypt '02, LNCS 2332, pages 299–320. Springer-Verlag, 2002.
11. O. Goldreich, S. Goldwasser, S. Halevi, *Public-key Cryptography from Lattice Reduction Problems*, in Proc. of Crypto '97, LNCS 1294, pages 112–131. Springer-Verlag, 1997.
12. J. Hoffstein, N. Howgrave-Graham, J. Pipher, J.H. Silverman, W. Whyte, *NTRUSign: Digital Signatures Using the NTRU Lattice*, December, 2001. Available from `http://www.ntru.com`.
13. J. Hoffstein, B.S. Kaliski, D. Lieman, M.J.B. Robshaw, Y.L. Yin, *Secure user identification based on constrained polynomials*, US Patent 6,076,163, June 13, 2000.
14. J. Hoffstein, D. Lieman, J.H. Silverman, *Polynomial Rings and Efficient Public Key Authentication*, in Proc. International Workshop on Cryptographic Techniques and E-Commerce (CrypTEC '99), Hong Kong, (M. Blum and C.H. Lee, eds.), City University of Hong Kong Press.
15. J. Hoffstein, J. Pipher, J.H. Silverman. *Enhanced encoding and verification methods for the NTRU signature scheme (ver. 2)*, May 30, 2001. Available from `http://www.ntru.com`.
16. J. Hoffstein, J. Pipher, J.H. Silverman, *NSS: The NTRU Signature Scheme*, preprint, November 2000. Available from `http://www.ntru.com`.
17. J. Hoffstein, J. Pipher, J.H. Silverman, *NSS: The NTRU Signature Scheme*, in Proc. of Eurocrypt '01, LNCS 2045, pages 211–228. Springer-Verlag, 2001.

18. J. Hoffstein, J. Pipher, J.H. Silverman, *NSS: The NTRU Signature Scheme: Theory and Practice*, preprint, 2001. Available from `http://www.ntru.com`.
19. J. Hoffstein, J. Pipher and J.H. Silverman, *NTRU: A New High Speed Public Key Cryptosystem*, in Proc. of Algorithm Number Theory (ANTS III), LNCS 1423, pages 267–288. Springer-Verlag, 1998.
20. A.K. Lenstra, H.W. Lenstra Jr., L. Lovász, *Factoring Polynomials with Rational Coefficients*, Mathematische Ann. 261 (1982), 513–534.
21. D. Micciancio, *The Shortest Vector in a Lattice is Hard to Approximate to within Some Constant*, in Proc. 39th Symposium on Foundations of Computer Science, 1998, 92–98.
22. P. Nguyen, *Cryptanalysis of the Goldreich-Goldwasser-Halevi Cryptosystem from Crypto '97*, 1999
23. P. Nguyen and J. Stern, *Lattice Reduction in Cryptology: An Update*, in Proc. of Algorithm Number Theory (ANTS IV), LNCS 1838, pages 85–112. Springer-Verlag, 2000.
24. C.-P. Schnorr, *A Hierarchy of Polynomial Time Lattice Basis Reduction Algorithms*, Theoretical Computer Science 53 (1987), 201–224.
25. J.H. Silverman, *Estimated Breaking Times for NTRU Lattices*, NTRU Technical Note #012, March 1999. Available from `http://www.ntru.com`.
26. L. Washington, *Introduction to Cyclotomic Fields*, Graduate Texts in Mathematics 83, 1982.
27. Consortium for Efficient Embedded Security. Efficient Embedded Security Standard (EESS) # 1: Draft 3.0. Available from `http://www.ceesstandards.org`.

A Proof of Proposition 1

As in Proposition 1, let $G = UU^T$ be the Gram matrix of an integral unitary basis matrix U, and let L_G be the associated hypercubic lattice. By definition, every hypercubic lattice is isomorphic to $\mathbb{Z}^N$, so has a Gram matrix of this form. The three problems to be shown equivalent are:
(A) Given G, find the shortest vectors in L_G.
(B) Given G, recover U, up to sign and order of the coordinates.
(C) Given G, construct an embedding $L_G \to \mathbb{Z}^N$.
Proof: **(A)⇒(B)**: To know the shortest vectors in terms of the basis represented by G is to have of a unitary V such that VGV^T is the identity. Then $G = V^{-1}V^{-T}$, so V^{-1} a is solution to $G = UU^T$. As VU is orthogonal, V^{-1} recovers U up to sign and order of the coordinates. **(B)⇒(C)**: Given any U such that $G = UU^T$, the rows of U embed L_G into $\mathbb{Z}^N$. **(C)⇒(A)**: Since $det(G) = 1$, the embedding, $L_G \to \mathbb{Z}^N$ is surjective. By Gaussian elimination, we can find $\mathbf{e_i}$ in terms of the basis defined by G, thus the shortest vectors of L_G.

B Theta Series

The theta series, of a lattice is a lattice *invariant*, of interest as a tool to distinguish non-isomorphic lattices[3]. Given a lattice L, and an integer m, define

[3] Non-isomorphic lattices may have identical theta functions.

N_m to be the number of lattice points of length squared m. Traditionally, these lattice invariants are collected into an element of the power series ring $\mathbb{Z}[[q]]$ as follows

$$\Theta(L,q) = 1 + \Sigma_{i=0}^{\infty} N_m q^m. \tag{3}$$

A convenient effect of this packaging is the relationship among theta series for lattices with an orthogonal decomposition (direct sum). Suppose $L_3 = L_1 \oplus L_2$. Then suppressing q,

$$\Theta(L_3) = \Theta(L_1) * \Theta(L_2). \tag{4}$$

As examples, $\theta(\mathbb{Z}) = 1 + 2q + 2q^4 + 2q^9 \ldots$, and by Eq.4,

$$\theta(\mathbb{Z}^N) = (1 + 2q + 2q^4 + 2q^9 \ldots)^N.$$

We are interested in the theta series as a means of understanding how difficult it is to realize a lattice distinguishing oracle. For the purposes of this paper, we can focus on the "Test" Gram matrices introduced in Section 3, since these are the only one used with the lattice distinguishing oracle.

These lattices are particularly simple, so we can easily compute the theta series by hand. Consider the lattices $Test_i$ used to compute $\lambda_1(\mathbf{v})$. There are N such lattices, one for each $i \in \{1, 2, \ldots, N\}$. As subsets of $\mathbb{Z}^N$, these lattices are spanned by the rows of twice the identity matrix, and one other vector, with exactly i odd coordinates. We calculate this theta series by adding together two power series. The first, representing vectors with all even coordinates is:

$$\theta(2\mathbb{Z}^N) = (1 + 2q^{2*2} + 2q^{4*4} + 2q^{6*6} \ldots)^N. \tag{5}$$

The second series, θ_2 ,represents vectors with i odd coordinates:

$$\theta_2 = (2q^1 + 2q^{3*3} + 2q^{5*5} \ldots)^i \cdot (1 + 2q^{2*2} + 2q^{4*4} + 2q^{6*6} \ldots)^{N-i}. \tag{6}$$

The second summand of the theta series of this lattices clearly depends on i. For example, if $i = \pm 1 \pmod 4$, then the shortest vector of odd squared length vector has squared length i. We can multiply two power series products above to any degree of precision and thus obtain N_m, the number of vectors of length squared m.

Similar calculations for the lattices involved in the computation of $\lambda_1(\mathbf{v})$, $k \geq 2$ are similarly easy. These calculations provide interesting statistics which measure ways in which the lattices compared by the LDO differ.