A introduction to lattice trapdoor

Ganyu Xu (g66xu)

Fall, 2023

1 Introduction

Trapdoor functions are the foundations of public-key cryptography. The most famous and popular trapdoor functions include the RSA trapdoor (it is easy to compute m^e from m and e but difficult to recover m from m^e and e) and discrete log (it is easy to compute g^x from g, x but hard to recover x from g^x, g). While these two trapdoors have been proved very successful over their decades of application, they are unfortunately both based on the hardness of integer factorization. Trapdoor functions based on other classes of hard problems have been of research interest since the early days of public-key cryptography, and they have gained increased attention as a result of the discovery of efficient quantum algorithm for integer factorization that threatens to break encryption and signature schemes based on RSA and Diffie-Hellman.

In this project is a survey a particular class of cryptographic trapdoor functions whose construction is based on lattices and whose security is based on the hardness of lattice problems such as the shortest vector problem (SVP) and the closest vector problem (CVP). Some preliminary mathematics surrounding lattice are introduced in chapter 2, followed by some additional discussion of hard lattice problems and known best algorithms to solve them in chapter 3. A lattice-based trapdoor, initially proposed by Goldreich-Goldwasser-Halevi in 1997, is described and its applications discussed in chapter 4. In chapter 5 we will discuss an improvement due to Micciancio that uses Hermite normal form to reduce key sizes while achieving optimal security.

2 Preliminaries

Throughout this survey, we denote vectors by bolded letters $\mathbf{v} \in \mathbb{Z}^n$. While lattices are defined as discrete subgroups over the real, in practice it is easier to exclusively work with integer lattices $\mathcal{L} \subseteq \mathbb{Z}^n$. We also exclusively work with full-rank lattices (whose basis $B \in \mathbb{Z}^{n \times n}$ is non-singular full-rank matrix).

There are a few preliminaries in the mathematics of lattices that are tremendously helpful with understanding the difficulty of hard lattice problems and with the trapdoor constructions based on these hard problems. We will discuss them in this chapter.

2.1 Unimodular relationship

First, we describe how two different basis of the same lattice relate to each other. This relationship plays a crucial role in the construction of the GGH trapdoor because the trapdoor scheme first generates a secret basis, then uses this relationship to generate a public basis of the same lattice.

¿ Let $\mathcal{L}(B)$ the lattice generated by the column vectors of matrix $B \in \mathbb{Z}^{n\times n}$, then $\mathcal{L}(B_1) = \mathcal{L}(B_2)$ if and only if there exists a unimodular matrix $U \in \mathbb{Z}^{n\times n}$ such that $B_1 = B_2U$

Here a unimodular matrix is a square matrix whose entries are integers and whose determinant is ± 1. In the forward direction, if B_1, B_2 generate the same lattice, then each column of B_1, B_2 is a lattice point and can be represented using the other basis, which means that there exists integer matrix U, V such that $B_1 = B_2U$ and $B_2 = B_1V$. With simple substitution, we find that $B_1 = B_2U = (B_1V)U$, so $B_1(I - VU) = 1$. Because B_1 is non-singular, it necessarily follows that $VU = 1$. By a symmetric argument, we also have $UV = 1$, which means that $V = U^{-1}$. we know U, V to both be integer matrix, so their determinants both necessarily have to be 1, thus proving that U, V to both be unimodular matrix.

In the backward direction, if $B_1 = B_2U$ for some unimodular matrix U, then columns of B_1 are already points in the lattice generated by B_2, meaning that $\mathcal{L}(B_1) \subseteq \mathcal{L}(B_2)$. Because U is unimodular, we also hve $B_2 = B_1U^{-1}$, where U^{-1} is an integer matrix (in fact U^{-1} is also unimodular), so by a symmetric argument we also have $\mathcal{L}(B_2) \subseteq \mathcal{L}(B_1)$. Therefore, $\mathcal{L}(B_1) = \mathcal{L}(B_2)$. ■

Knowing that two basis generate the same lattice if and only if they are related by a unimodular matrix, it is not hard to deduce that swapping columns, negating a column, and adding to a column an integer linear combination of all other columns all preserve the lattice generated by a matrix because each of the such action can be represented as a right multiplication by a unimodular matrix.

2.2 Orthogonality and determinant

The determinant of a lattice $\mathcal{L}(B)$ generated by a basis B is the determinant of the matrix B. Knowing that two basis generate the same lattice if and only if they are related by a unimodular matrix (whose determinant is 1), it is not hard to see that the determinant is the same regardless of choice of basis. We can thus define the "determinant of a lattice" $\det(\mathcal{L})$ as an invariant of the lattice regardless of the choice of basis.

The determinant of a lattice can be geometrically interpreted as the "n-dimensional volume" of the fundamental parallelpiped, which is defined as a subset of $\mathbb{R}^n$:

$$P(B) = B \cdot [0,1)^n = \{\sum_{i=1}^{n} x_i \mathbf{b_i} \mid 0 \leq x < 1\}$$

Note that it is often more convenient to "center" the fundamental parallelpiped around $\mathbf{0}$ and define it as $P(B) = B \cdot [-\frac{1}{2}, \frac{1}{2})^n$.

On the other hand, we are also concerned with "how orthogonal" the basis is. For that we define the orthogonality defect:

$$\text{ortho-defect} = \frac{\prod_{i=1}^{n} \|\mathbf{b}_i\|}{|\det(B)|}$$

Observe that if B is also an orthogonal matrix whose columns are pair-wise orthogonal, then ortho-defect$(B) = 1$. Otherwise, the orthogonality defect of B will be greater than 1. Geometrically, a greater orthogonality defect corresponds to a basis that is more "skewed".

Intuitively it is not hard to imagine that, for a fundamental parallelpiped with a fixed n-dimensional volume, the more skewed the basis is, the longer each basis will have to be to preserve the n-dimensional volume, and the further away the basis is from the origin. This is the principal observation on the hardness of the shortset vector problems and their siblings (such as the closest vector problem used in GGH's construction): **the longer, more skewed a basis is, the harder it is to find a short vector in the lattice**. It turns out that the gap in difficult of finding short vector between good (short, orthogonal) basis and bad (long, skewed) basis is exponential, and thus hard lattice problems pose provide an excellent foundation to build trapdoor functions on.

‘’

3 Lattice basis reduction

In this section we discuss some known best bounds on the shortest vector (of a lattice), as well as the known best algorithm for approximating the shortest vector.

By convention, we define the minimum distance of a lattice to be the minimum of all distances between two distinct points on a lattice. Because lattice is a discrete sub-group of $\mathbb{Z}^n$, this minimum distance is always achieved by some pair of points. Also because a lattice is closed under addition (and thus subtraction), the minimum distance is exactly the length of the shortest vector on the lattice. Denote the mininum distance by $\lambda(\mathcal{L})$. For convenience, where there is no confusion of which lattice we are working with, we simply denote the minimum distance by λ

3.1 A lower bound on the shortest vector

We can first provide a lower bound on the length of the shortest vector of a lattice, given a basis B. First, recall the Gram-Schmidt orthognoalization process for converting a basis $B = [\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n]$ into an orthogonal basis $B^* = [\mathbf{b}_1^*, \mathbf{b}_2^*, \ldots, \mathbf{b}_n^*]$ via the following algorithm:

$$\mathbf{b}_i^* \leftarrow \mathbf{b}_i - \sum_{j<i} \frac{\langle \mathbf{b}_i, \mathbf{b}_j^* \rangle}{\langle \mathbf{b}_j^*, \mathbf{b}_j^* \rangle} \mathbf{b}_j^*$$

For convenience, we denote the orthogonalization coefficient by:

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

Also, we define a function π_i that projects a vector from $\mathbb{R}^n$ onto the orthogonal basis formed by $\mathbf{b}_i^*, \mathbf{b}_{i+1}^*, \ldots, \mathbf{b}_n^*$:

$$\pi_i(\mathbf{x}) = \sum_{j \geq i} \frac{\langle \mathbf{x}, \mathbf{b}_j^* \rangle}{\langle \mathbf{b}_j^*, \mathbf{b}_j^* \rangle} \mathbf{b}_j^*$$

Observe that for $i = 1$, π_i is the identity function, because the orthogonalized basis still span the same vector space as the original basis. Also, $\pi_i(\mathbf{b}_i) = \mathbf{b}_i^*$:

$$\begin{aligned}
\pi_i(\mathbf{b}_i) &= \sum_{j \geq i} \frac{\langle \mathbf{b}_i, \mathbf{b}_j^* \rangle}{\langle \mathbf{b}_j^*, \mathbf{b}_j^* \rangle} \mathbf{b}_j^* \\
&= \sum_{1 \leq j \leq n} \frac{\langle \mathbf{b}_i, \mathbf{b}_j^* \rangle}{\langle \mathbf{b}_j^*, \mathbf{b}_j^* \rangle} \mathbf{b}_j^* - \sum_{j < i} \frac{\langle \mathbf{b}_i, \mathbf{b}_j^* \rangle}{\langle \mathbf{b}_j^*, \mathbf{b}_j^* \rangle} \mathbf{b}_j^* \\
&= \pi_1(\mathbf{b}_i) - \sum_{j < i} \frac{\langle \mathbf{b}_i, \mathbf{b}_j^* \rangle}{\langle \mathbf{b}_j^*, \mathbf{b}_j^* \rangle} \mathbf{b}_j^* \\
&= \mathbf{b}_i - \sum_{j < i} \frac{\langle \mathbf{b}_i, \mathbf{b}_j^* \rangle}{\langle \mathbf{b}_j^*, \mathbf{b}_j^* \rangle} \mathbf{b}_j^* \\
&= \mathbf{b}_i^*
\end{aligned}$$

With these notations established, we can bound $\lambda(\mathcal{L}(\mathcal{B}))$ from below using the orthogonalized basis. We will do that by proving the following recursive inequality:

$$\lambda(\mathcal{L}(\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n)) \geq \min(\|\mathbf{b}_n^*\|, \lambda(\mathcal{L}(\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_{n-1}))$$

To prove this inequality, we represent an arbitrary non-zero point on the lattice by $B\mathbf{x} = \sum_{i=1}^n \mathbf{b}_i x_i$:

$$\begin{aligned}
\|B\mathbf{x}\|^2 &= \|\sum_{i=1}^{n} \mathbf{b}_i x_i\|^2 \\
&= \|\sum_{i=1}^{n-1} \mathbf{b}_i x_i + \mathbf{b}_n x_n\|^2
\end{aligned}$$

From here substitute the orthogonalization of $\mathbf{b}_n^* = \mathbf{b}_n - \sum_{j<n} \mu_{j,n} \mathbf{b}_j^*$:

$$\begin{aligned}
\|B\mathbf{x}\|^2 &= \|\sum_{i=1}^{n-1} \mathbf{b}_i x_i + \mathbf{b}_n x_n\|^2 \\
&= \|\sum_{i=1}^{n-1} \mathbf{b}_i x_i + (\mathbf{b}_n^* + \sum_{j<n} \mu_{j,n} \mathbf{b}_j^*) x_n\|^2 \\
&= \|\sum_{i=1}^{n-1} \mathbf{b}_i x_i + (\sum_{j<n} \mu_{j,n} \mathbf{b}_j^*) x_n + \mathbf{b}_n^* x_n\|^2
\end{aligned}$$

Observe that by the definition of the orthogonalization process, $\mathbf{b}_n^*$ is orthogonal to all other orthogonal basis $\mathbf{b}_1^*, \mathbf{b}_2^*, \ldots, \mathbf{b}_{n-1}^*$, as well as all except the identically indexed original basis $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_{n-1}$. By Pythagoras theorem we can separate the RHS into orthogonal components:

$$\begin{aligned}
\|B\mathbf{x}\|^2 &= \|\sum_{i=1}^{n-1} \mathbf{b}_i x_i + (\sum_{j<n} \mu_{j,n} \mathbf{b}_j^*) x_n + \mathbf{b}_n^* x_n\|^2 \\
&= \|\sum_{i=1}^{n-1} \mathbf{b}_i x_i + (\sum_{j<n} \mu_{j,n} \mathbf{b}_j^*) x_n\|^2 + \|\mathbf{b}_n^* x_n\|^2 \\
&\geq \|\mathbf{b}_n^* x_n\|^2
\end{aligned}$$

where $x_n > 0$, the inequality trivially implies $\|B\mathbf{x}\| \geq \|\mathbf{b}_n^*\|$. where $x_n = 0$, $\mathcal{L}(\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n) = \mathcal{L}(\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_{n-1})$, so by definition of minimum distance we have $\|B\mathbf{x}\| \geq \lambda(\mathcal{L}(\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_{n-1}))$. Combining the two cases gives us the overall inequality. ■

Having proved the recursive inequality, we can simply apply recursion n times and arrive at a concrete lower bound for λ.

$$\lambda \geq \min_{1 \leq i \leq n} \|\mathbf{b}_i^*\|$$

3.2 Reduced lattice basis

At the time of this survey, the best algorithm for solving the (approximate) shortest vector problem is the LLL lattice basis reduction algorithm, attributed to Arjen Lenstra, Hendrik Lenstra, and László Lovász. The reduction algorithm transforms an input basis into an LLL-reduced form, where the first base vector of the reduced basis is an approximation of the shortest vector. Details of the actual algorithm for obtaining a reduced basis will be discussed in section 4. Meanwhile, it is helpful to first discuss the properties of the reduced form itself.

The definition of a reduced basis is parameterized by a real number $\frac{1}{4} < \delta \leq 1$. A basis B is δ-LLL reduced if the following conditions are satisfied:

1. For all $j > i$, $|\mu_{j,i}| \leq \frac{1}{2}$
2. For all $1 \leq i \leq n-1$, $\delta \|\pi_i(\mathbf{b}_i)\|^2 \leq \|\pi_i(\mathbf{b}_{i+1})\|^2$

The second clause of the reduced form is of particular interest because we can use it to measure how well $\mathbf{b}_1$ approximates the shortest vector. First observe the RHS of the condition:

$$\begin{aligned}
\|\pi_i(\mathbf{b}_{i+1})\|^2 &= \|\sum_{j \geq i} \mu_{i+1,j} \mathbf{b}_j^*\|^2 \\
&= \|\mu_{i+1,i} \mathbf{b}_i^* + \sum_{j \geq i+1} \mu_{i+1,j} \mathbf{b}_j^*\|^2 \\
&= \|\mu_{i+1,i} \mathbf{b}_i^* + \pi_{i+1}(\mathbf{b}_{i+1})\|^2 \\
&= \|\mu_{i+1,i} \mathbf{b}_i^* + \mathbf{b}_{i+1}^*\|^2 \\
&= \|\mu_{i+1,i} \mathbf{b}_i^*\|^2 + \|\mathbf{b}_{i+1}^*\|^2
\end{aligned}$$

Substituting the equation back into the condition:

$$\delta\|\pi_i(\mathbf{b}_i)\|^2 \leq \|\mu_{i+1,i}\mathbf{b}_i^*\|^2 + \|\mathbf{b}_{i+1}^*\|^2$$

Which transforms into:

$$\|\mathbf{b}_i^*\|^2 \leq \frac{1}{(\delta - \mu_{i+1,i}^2)}\|\mathbf{b}_{i+1}^*\|^2$$

By the first clause of the reduced basis we know $\mu_{i+1,i}^2 \leq \frac{1}{4}$, which means that $\frac{1}{\delta-\mu_{i+1,i}^2} \leq \frac{1}{\delta-\frac{1}{4}}$. Denote $\alpha = \frac{1}{\delta-\frac{1}{4}}$, then $\alpha > \frac{4}{3}$, and we have the following inequality:

$$\|\mathbf{b}_i^*\|^2 \leq \alpha\|\mathbf{b}_{i+1}^*\|^2$$

Note that because $\mathbf{b}_1 = \mathbf{b}_1^*$, we can recusively evaluate the inequality above can obtain the following closed inequality:

$$\|\mathbf{b}_1\|^2 \leq \alpha^{i-1}\|\mathbf{b}_i^*\|^2 \leq \alpha^{n-1}\|\mathbf{b}_i^*\|^2$$

The inequality above states that $\mathbf{b}_1$ is not longer than $\alpha^{n-1}\|\mathbf{b}_i^*\|$ for all i, so it must be not longer than the minimum among $\alpha^{n-1}\|\mathbf{b}_i^*\|$:

$$\|\mathbf{b}_1\|^2 \leq \min_{1\leq i\leq n} \alpha^{n-1}\|\mathbf{b}_i^*\|^2 = \alpha^{n-1}\min_{1\leq i\leq n}\|\mathbf{b}_i^*\|^2$$

From the previou section we've derived that $\lambda \geq \min_{1\leq i\leq n}\|\mathbf{b}_i^*\|$, which we can plug into the inequality above:

$$\|\mathbf{b}_1\| \leq \alpha^{\frac{n-1}{2}}\lambda$$

This in equality bounds the length of the first base vector of the reduced basis by some multiples of the minimum distance, where the multiple is exponential with respect to the number of dimensions of the lattice.

In other words, the LLL reduction algorithm computes an approximation of the shortest vector within a factor of $\gamma \in O(\alpha^n)$. With specific choices of "bad" basis and sufficiently high dimension n, the basis reduction algorithm will not be able to provide meaningfully tight approximation of the shortest vector, making the shortest vector problem suitably hard for cryptographic applications.

3.3 Fundamental regions

A fundamental region S is a subset of the (real) linear span of the basis such that it tiles the linear span of the basis and each tile contains exactly one lattice point. If we have a fundamental region S defined, then each point in the linear span of the basis can be uniquely decomposed into the sum of a lattice point and a point in the fundamental region:

$$\mathcal{L}(B) = \{S + B\mathbf{x} \mid \mathbf{x} \in \mathbb{Z}^n\}$$

A few notable examples include the fundamental parallelpiped

$$\mathcal{P}(B) = \{B\mathbf{x} \mid \mathbf{x} \in [0,1)^n\}$$

and the centered fundamental parallelpiped

$$\mathcal{C}(B) = \{B\mathbf{x} \mid \mathbf{x} \in [-\frac{1}{2}, \frac{1}{2})^n\}$$

Computing the decomposition using the (centered) fundamental parallelpiped is easy. First compute $B^{-1}\mathbf{t}$ (over $\mathbb{R}$), then perform some kind of rounding (flooring or nearest integer) depending on whether the fundamental parallelpiped is centered or or not. $B\lfloor B^{-1}\mathbf{T}\rceil$ is the lattice point whose corresponding fundamental region contains the target. This is Babai's rounding algorithm. As we will see in later section, this algorithm works well if B is short and orthogonal, but falls apart badly if B is long and skewed.

The Voronoi region is defined by the set of points in the linear span that are closer to $\mathbf{0}$ than to any other lattice points. There is some additional details that need to be specified to ensure that the Voronoi region properly closes and forms a partition of span(B), which we will not discuss in details here. If we have an algorithm that can efficiently decompose $\mathbf{t} \in \text{span}(B)$ using the Voronoi region, then we automatically have a way to solve the closest vector problem. Unfortunately, no such algorithm is known.

Of particular interest is the (centered) orthogonalized fundamental parallelpiped, which is defined using the orthogonalized basis $B^* = \text{Gram-Schmidt}(B)$:

$$\mathcal{C}(B^*) = \{B^*\mathbf{x} \mid \mathbf{x} \in [-\frac{1}{2}, \frac{1}{2})\}$$

(If time permits we should try to prove that $\mathcal{C}(B^*)$ is indeed a fundamental region. For now, we will take that for granted without proof).

Because B^* is an orthogonal basis, it's easy to see that the sphere centered at some lattice point $\mathbf{v} \in \mathcal{L}$ and whose radius is $\frac{1}{2}\min \|\mathbf{b}_i^*\|$ is entirely contained in the (shifted) fundamental region sphere $\subset \{\mathcal{C}(B^*) + \mathbf{v}\}$.

3.4 Babai's nearest plane algorithm

Babai's nearest plane algorithm, attributed to László Babai, is a recursive algorithm that can approximate the closest vector under a given basis to a target vector. More specifically, it returns a vector point $\mathbf{v}$ such that, if target vector is projected onto the orthogonalized basis, the projection is contained in $\mathbf{v} + \mathcal{C}(B^*)$. If the target vector is in the linear span of the basis, then the target vector itself is contained in $\mathbf{v} + \mathcal{C}(B^*)$.

Algorithm 1 NearestPlane

```
1: if $B$ is empty then
2:     return $\mathbf{0}$
3: end if
4: $B^* \leftarrow$ GramSchmidt($B$)
5: $c \leftarrow \lfloor \frac{\langle \mathbf{t}, \mathbf{b}_n^* \rangle}{\langle \mathbf{b}_n^*, \mathbf{b}_n^* \rangle} \rceil$, where $\mathbf{b}_n^*$ is the last base vector in $B^*$
6: return $c\mathbf{b}_n$ + NearestPlane($[\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_{n-1}], \mathbf{t} - c\mathbf{b}_n$)
```

Denote the output of this algorithm by $\mathbf{v}$, the key theorem about this algorithm is as follows:

$$\forall 1 \leq i \leq n, \frac{\langle \mathbf{t} - \mathbf{v}, \mathbf{b}_i^* \rangle}{\langle \mathbf{b}_i^*, \mathbf{b}_i^* \rangle} \in [-\frac{1}{2}, \frac{1}{2})$$

Intuitively, the inequality above states that the deviation of $\mathbf{t}$ from $\mathbf{v}$ is between $-\frac{1}{2}\mathbf{b}_i^*$ and $\frac{1}{2}\mathbf{b}_i^*$ in any of the chosen direction $\mathbf{b}_i^*$, which means that $\mathbf{t} - \mathbf{v}$ is indeed contained in the orthogonalized fundamental parallelpiped.

We can prove this result inductively. In the base case, if the input set of basis is empty, then this result is trivially correct. In the inductive case, we denote the output of NearestPlane$(B', \mathbf{t} - c\mathbf{b}_n)$ by $\mathbf{v}'$. Assuming that the algorithm produces the desired result for the sublattice $\mathcal{L}(B')$:

$$\forall 1 \leq i \leq (n-1), \frac{\langle (\mathbf{t} - c\mathbf{b}_n) - \mathbf{v}', \mathbf{b}_i^* \rangle}{\langle \mathbf{b}_i^*, \mathbf{b}_i^* \rangle} \in [-\frac{1}{2}, \frac{1}{2})$$

However, notice in step 5 of the algorithm $c\mathbf{b}_n + \mathbf{v}'$ is the output of the current iteration of the algorithm, so we have

$$\begin{aligned}(\mathbf{t} - c\mathbf{b}_n) - \mathbf{v}' &= \mathbf{t} - (c\mathbf{b}_n + \mathbf{v}') \\ &= \mathbf{t} - \mathbf{v}\end{aligned}$$

which means that

$$\forall 1 \leq i \leq (n-1), \frac{\langle \mathbf{t} - \mathbf{v}, \mathbf{b}_i^* \rangle}{\langle \mathbf{b}_i^*, \mathbf{b}_i^* \rangle} \in [-\frac{1}{2}, \frac{1}{2})$$

So all that remains is to show that this relationship also holds for $i = n$.

To prove that the relationship holds for $i = n$, first observe how c is computed:

$$c \leftarrow \lfloor \frac{\langle \mathbf{t}, \mathbf{b}_n^* \rangle}{\langle \mathbf{b}_n^*, \mathbf{b}_n^* \rangle} \rceil$$

which is equivalent to saying that

$$\frac{\langle \mathbf{t}, \mathbf{b}_n^* \rangle}{\langle \mathbf{b}_n^*, \mathbf{b}_n^* \rangle} - c \in [-\frac{1}{2}, \frac{1}{2})$$

Also recall the Gram-Schmidt orthogonalization process:

$$\mathbf{b}_n^* + \sum_{i<n} \mu_{n,i} \mathbf{b}_i^* = \mathbf{b}_n$$

Therefore

$$\begin{aligned}\langle \mathbf{b}_n, \mathbf{b}_n^* \rangle &= \langle \mathbf{b}_n^* + \sum_{i<n} \mu_{n,i} \mathbf{b}_i^*, \mathbf{b}_n^* \rangle \\ &= \langle \mathbf{b}_n^*, \mathbf{b}_n^* \rangle\end{aligned}$$

All non $\mathbf{b}_n^*$ terms can be cleared because they are orthogonal to $\mathbf{b}_n^*$, so their inner product is 0. This equality means that:

$$\frac{\langle \mathbf{b}_n, \mathbf{b}_n^* \rangle}{\langle \mathbf{b}_n^*, \mathbf{b}_n^* \rangle} = 1$$

Finally we can put everything together:

$$\begin{aligned}\frac{\langle \mathbf{t}, \mathbf{b}_n^* \rangle}{\langle \mathbf{b}_n^*, \mathbf{b}_n^* \rangle} - c &= \frac{\langle \mathbf{t}, \mathbf{b}_n^* \rangle}{\langle \mathbf{b}_n^*, \mathbf{b}_n^* \rangle} - c\frac{\langle \mathbf{b}_n, \mathbf{b}_n^* \rangle}{\langle \mathbf{b}_n^*, \mathbf{b}_n^* \rangle} \\ &= \frac{\langle \mathbf{t} - c\mathbf{b}_n, \mathbf{b}_n^* \rangle}{\langle \mathbf{b}_n^*, \mathbf{b}_n^* \rangle} \in [-\frac{1}{2}, \frac{1}{2})\end{aligned}$$

Hence we've proved the relationship for $i = n$. ■

3.5 Basis size reduction

In addition, to being the best known algorithm for solving the CVP problem, Babai's nearest plane algorithm is also an essential component to the LLL lattice basis reduction algorithm. Specifically, the nearest plane algorithm is used to reduce the size of the basis vector so the first condition of the reduced basis (shown below) can be satisfied:

$$\forall j < i, |\mu_{i,j}| \leq \frac{1}{2}$$

The size reduction algorithm, which we will denote by SizeReduce, takes a basis B and returns a size-reduced basis B' such that the condition above is true. The procedure for SizeReduce is as follows:

Algorithm 2 SizeReduce

```
1: for $i \in \{2, 2, \ldots, n\}$ do
2:     $\mathbf{v} \in \mathcal{L}([\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}]) \leftarrow$ NearestPlane($[\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}], \mathbf{b}_i - \mathbf{b}_i^*$)
3:     $\mathbf{b}_i \leftarrow \mathbf{b}_i - \mathbf{v}$
4: end for
```

Recall the result of the nearest plane algorithm we know that:

$$\forall j < i, \frac{\langle \mathbf{b}_i - \mathbf{b}_i^* - \mathbf{v}, \mathbf{b}_j^* \rangle}{\langle \mathbf{b}_j^*, \mathbf{b}_j^* \rangle} \in [-\frac{1}{2}, \frac{1}{2})$$

Notice in the equation above, we have $\mathbf{b}_i^* \perp \mathbf{b}_j^*$ because $j < i$. We also have $\mathbf{b}_i - \mathbf{v}$ being the new value for $\mathbf{v}_i$ after the substitution. This means that after the substitution:

$$\forall j < i, \frac{\langle \mathbf{b}_i, \mathbf{b}_j^* \rangle}{\langle \mathbf{b}_j^*, \mathbf{b}_j^* \rangle} \in [-\frac{1}{2}, \frac{1}{2})$$

The LHS of the equation above is exactly $\mu_{i,j}$. Thus we have satisfied the first condition of δ-LLL reduced basis. In addition, because $\mathbf{v} \in \mathcal{L}([\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}])$, the substitution $\mathbf{b}_i \leftarrow \mathbf{b}_i - \mathbf{v}$ is equivalent to adding onto the substituted basis a linear combination of other basis vectors, which corresponds to a unimodular matrix multiplication and thus does not change the lattice itself before or after the size reduction.

3.6 The LLL basis reduction algorithm

Now that we have the SizeReduce algorithm which takes a basis and transforms it into a size-reduced basis for the same lattice that satisfies the first condition of a δ-LLL reduced basis, it remains somehow transform the basis to satisfy the second condition:

$$\delta\|\pi_i(\mathbf{b}_i)\|^2 \leq \|\pi_i(\mathbf{b}_{i+1})\|^2$$

It turns out that such transformation is rather simple: if there is some i such that the condition above does not hold, then we can simply swap $\mathbf{b}_i$ and $\mathbf{b}_{i+1}$, and the condition will hold. To see that it works, denote the swapped basis vectors by $\mathbf{b}'_i = \mathbf{b}_{i+1}$, $\mathbf{b}'_{i+1} = \mathbf{b}_i$. First observe that the function $\pi_i : \mathbf{x} \mapsto \sum_{j \geq i} \frac{\langle \mathbf{x}, \mathbf{b}^*_j \rangle}{\langle \mathbf{b}^*_j, \mathbf{b}^*_j \rangle} \mathbf{b}^*_j$ is not changed after the swap because it is still projecting $\mathbf{x}$ onto the same set of orthogonal basis, and the set of orthogonal basis is unchanged by the swap.

if the condition does not hold, then $\delta\|\pi_i(\mathbf{b}_i)\|^2 > \|\pi_i(\mathbf{b}_{i+1})\|^2$, and we have

$$\begin{aligned}
\delta\|\pi_i(\mathbf{b}'_i)\|^2 &= \delta\|\pi_i(\mathbf{b}_{i+1})\|^2 \\
&< \delta \cdot \delta\|\pi_i(\mathbf{b}_i)\|^2 \\
&\leq \|\pi_i(\mathbf{b}_i)\|^2 \\
&= \|\pi_i(\mathbf{b}'_{i+1})\|^2
\end{aligned}$$

So after the swap, the condition will hold. We know that swapping columns preserves the lattice, so we can define a second algorithm ColumnSwap that takes a basis B and transforms it into a second basis of the same lattice that satisfies the second condition of δ-LLL reduced basis:

Algorithm 3 ColumnSwap

```
1: for $i \in \{1, 2, \ldots, n-1\}$ do
2:     if $\delta\|\pi_i(\mathbf{b}_i)\|^2 > \|\pi_i(\mathbf{b}_{i+1})\|^2$ then
3:         $\mathbf{b}_i \leftarrow \mathbf{b}_{i+1}$
4:         $\mathbf{b}_{i+1} \leftarrow \mathbf{b}_i$
5:     end if
6: end for
```

Unfortunately, after applying ColumnSwap, the first condition no longer holds, so we will need to apply SizeReduce again. In fact, the LLL basis reduction algorithm is exactly repeatedly alternate between ColumnSwap and SizeReduce.

Algorithm 4 LLLReduce

```
1: while $B$ is not $\delta$-LLL reduced do
2:     $B \leftarrow$ SizeReduce($B$)
3:     $B \leftarrow$ ColumnSwap($B$)
4: end while
```

We can easily see that if LLLReduce terminates, we will have a correctly reduced basis. It remains to show that LLLReduce will terminate in polynomial time (at least for when $\delta < 1$.

To prove that LLLReduce will terminate in polynomial time, we define a positive integer quantity associated with a basis and show that each iteration of SizeReduce and ColumnSwap reduce this quantity by δ.

Recall that although determinant is not defined for non-square matrix, it is defined for (sub)lattice generated by sub-basis $B_k = [\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_k] \in \mathbb{Z}^{n \times k}$ where $1 \leq k \leq n$:

$$\det(\mathcal{L}(B_k)) = \prod_{i=1}^{k} \|\mathbf{b}_i^*\| = \sqrt{B_k^{\intercal} B_k}$$

We define the "potential" of a full-rank basis $B = [\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n]$ by the product of determinants of sub-lattices generated by sub-basis $B_1, B_2, \ldots, B_n$:

$$\mathcal{D} = \prod_{k=1}^{n} \det(\mathcal{L}(B_k))^2$$

Based on the definition of lattice determinant above, we know that the square of the determinant of a lattice generated by integer basis is an integer, so their products must also be integer. Therefore, $\mathcal{D}$ is an integer.

First observe that in SizeReduce, the basis vector is changed by subtracting a linear combination of basis vectors BEFORE the changed basis vector. This means that after SizeReduce, each of B_k for $1 \leq k \leq n$ still generates the same lattice, and the determinant of that lattice remains unchanged. In other words, SizeReduce does not change the value of $\mathcal{D}$.

Second, for $k < i$, swapping column $\mathbf{b}_i$ with $\mathbf{b}_{i+1}$ does not affect the lattice generated by the partial basis B_k because $B_k = \{\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}\}$ does not contain the swapped columns anyways. On the other hand, for $k > i$, swapping column $\mathbf{b}_i$ with $\mathbf{b}_{i+1}$ also does not affect the lattice generated by the partial basis B_k because it contains BOTH of the swapped column, and swapping column preserves the lattice.

Therefore the only change to $\mathcal{D}$ when swapping $\mathbf{b}_i$ with $\mathbf{b}_{i+1}$ comes from the factor $\det(\mathcal{L}(B_i))$. Denote the potential of the basis B after the swap by $\mathcal{D}'$ then:

$$\frac{\mathcal{D}}{\mathcal{D}'} = \frac{\det(\mathcal{L}([\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_{i-1}, \mathbf{b}_i]))^2}{\det(\mathcal{L}([\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_{i-1}, \mathbf{b}_{i+1}]))^2}$$

Recall that the determinant of a lattice is the product of the orthogonalized basis vectors. The first $i-1$ terms are identical between the two basis so they all cancel out. The i-th term on the numerator is exactly $\mathbf{b}_i^*$, while the i-th term on the denominator is as follows:

$$\begin{aligned}
\text{i-th term in denominator} &= \mathbf{b}_{i+1} - \sum_{j<i} \mu_{i+1,j}\mathbf{b}_j^* \\
&= \mathbf{b}_{i+1} - \Big(\sum_{1\le j\le n} \mu_{i+1,j}\mathbf{b}_j^* - \sum_{j\ge i} \mu_{i+1,j}\mathbf{b}_j^*\Big) \\
&= \mathbf{b}_{i+1} - \Big(\mathbf{b}_{i+1} - \sum_{j\ge i} \mu_{i+1,j}\mathbf{b}_j^*\Big) \\
&= \sum_{j\ge i} \mu_{i+1,j}\mathbf{b}_j^* \\
&= \pi_i(\mathbf{b}_{i+1})
\end{aligned}$$

Therefore we have:

$$\frac{\mathcal{D}}{\mathcal{D}'} = \frac{\det(\mathcal{L}([\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_{i-1}, \mathbf{b}_i]))^2}{\det(\mathcal{L}([\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_{i-1}, \mathbf{b}_{i+1}]))^2} = \frac{\|\mathbf{b}_i^*\|^2}{\|\pi_i(\mathbf{b}_{i+1})\|^2} = \frac{\|\pi_i(\mathbf{b}_i)\|^2}{\|\pi_i(\mathbf{b}_{i+1})\|^2}$$

Because we only swap when $\delta\|\pi_i(\mathbf{b}_i)\|^2 > \|\pi_i(\mathbf{b}_{i+1})\|^2$, the quotient above satisfies:

$$\frac{\mathcal{D}}{\mathcal{D}'} = \frac{\|\pi_i(\mathbf{b}_i)\|^2}{\|\pi_i(\mathbf{b}_{i+1})\|^2} > \frac{1}{\delta}$$

Which means that $\mathcal{D}' < \delta\mathcal{D}$. $\blacksquare$

4 The Goldreich-Goldwasser-Halevi trapdoor

One class of constructing lattice trapdoor uses a pair of public and secret basis for the same lattice. Since the two basis generate the same lattice, they are equally good at mapping integer coordinates into a lattice point. On the other hand, the secret basis is very "good" and can be used to efficiently find the closest lattice point, but the public basis is very "bad" at recovering the closest lattice point.

One of the earliest instances of such class of lattice trapdoor is proposed by Goldreich, Goldwasser, and Halevi in their 1997 paper *"Public-key cryptosystem from lattice reduction problem"*. At a high level, the trapdoor is parameterized by three items: a "bad" basis B, a "good" basis R, and an error bound σ. In the forward direction, the function maps a pair of lattice coordinate $\mathbf{v} \in \mathbb{Z}^n$ and a small error vector $\mathbf{e} \leftarrow \{-\sigma, \sigma\}^n$ to $\mathbf{x} = B\mathbf{v} + \mathbf{e} \in \mathbb{R}^n$. If the parameters are generated correctly, then the closest lattice point in $\mathcal{L}(B)$ is exactly $B\mathbf{v}$.

Inverting the function involves recovering the integer coordinate $\mathbf{v}$ (or the error vector $\mathbf{e}$, since recovering one of them automatically gives you the other). However, the inversion is exactly the closest vector problem (CVP), and should be hard if B is a sufficiently bad basis. On the other hand, since R is a good basis, finding the closest vector point should be "easy" if we have R.

From here, GGH '97 proposed a public-key cryptosystem as well as a digital signature scheme that uses such a trapdoor construction, and the security of the two schemes naturally rest on the hardness of the underlying hard lattice problem.

4.1 The trapdoor scheme

The GGH trapdoor scheme contains four components:

1. **Parameter generation** The main security parameter in this scheme is the number of dimensions n of the lattice. In the original paper, the authors claimed $n \approx 150$ is sufficient for making inverting the function without the private basis hard and $n \approx 250$ should be a safe choice for the foreseeable future.

2. **Key generation** First generate the "good" basis $R \in \mathbb{R}^{n\times}$, then apply some (unimodular) transformation to R to obtain the "bad" basis B. The error bound $\sigma > 0 \in \mathbb{R}$ is dependent on the choice of R and the choice of "probability of inversion error" $\epsilon >= 0$, which will be discussed in a later section.

3. **Forward evaluation** $f_B(\mathbf{v}, \mathbf{e}) = B\mathbf{v} + \mathbf{e}$, where $\mathbf{v} \in \{-n, \ldots, n\}^n$ and $\mathbf{e} \in \{-\sigma, \sigma\}$. According to the authors, the choice of bounds for values of $\mathbf{v}$ is arbitrary and not a significant contributor to the overall security of the scheme.

4. **Inversion** Denote the output by $\mathbf{x} = f_B(\mathbf{v}, \mathbf{e})$, first attempt to recover the integer coordinate $\mathbf{v} \leftarrow B^{-1}R\lfloor R^{-1}\mathbf{x}\rceil$. From here it is trivial to recompute the lattice point $B\mathbf{v}$ and recover the error term $\mathbf{e} = \mathbf{x} - B\mathbf{v}$.

4.2 Correctness of trapdoor inversion

Without the error term, the function $f_B : \mathbf{v} \mapsto B\mathbf{v}$ is trivially invertible with either choice of the basis. However, with a non-zero the error term, the quality of the basis makes a substantial difference in how much error can be added before the points can no longer be recovered.

Observe the calculation used for recovering the integer coordinate $\mathbf{v}$:

$$\begin{aligned}\lfloor R^{-1}\mathbf{x}\rceil &= B^{-1}R\lfloor R^{-1}(B\mathbf{v} + \mathbf{e})\rceil \\ &= B^{-1}R\lfloor R^{-1}B\mathbf{v} + R^{-1}\mathbf{e}\rceil\end{aligned}$$

Since R, B are related by a unimodular matrix and $\mathbf{v}$ is an integer vector, $R^{-1}B\mathbf{v}$ is an integer vector and can be moved out of the "rounding" operator:

$$\begin{aligned}\lfloor R^{-1}\mathbf{x}\rceil &= B^{-1}RR^{-1}B\mathbf{v} + B^{-1}R\lfloor R^{-1}\mathbf{e}\rceil \\ &= \mathbf{v} + B^{-1}R\lfloor R^{-1}\mathbf{e}\rceil\end{aligned}$$

Since $B^{-1}R$ is also a unimodular matrix, we can conclude that the equation above is successful at recovering the original coordinate if and only if $R^{-1}\mathbf{e} = \mathbf{0}$.

To guarantee that inversion error never happens, we can bound the error term $\sigma > 0$ by $\frac{1}{2\rho}$, where ρ is maximal L_1 norm among the rows of R^{-1}. This bound is excessively conservative, however, and we might want to relax the bound to enhance the security of the trapdoor scheme (larger error terms makes it harder to invert the function using only the public basis). The authors provided one such relaxation based on the Hoeffding inequality. This relaxation is stated as follows

$$P(\text{inversion error}) \leq 2n \cdot \exp(-\frac{1}{8\sigma^2\gamma^2})$$

Where $\gamma = \sqrt{n} \cdot \max(L_\infty \text{ norm of rows of } R^{-1})$. Simple reorganization of inequality shows that to bound the inversion error by ϵ, the error term will be bounded by $\sigma \leq (\gamma\sqrt{8 \ln 2n/\epsilon})^{-1}$.

4.3 Generating the pair of basis

The authors described two similar ways of generating the private basis. The first way is to sample each coordinate of $R \in \mathbb{R}^{n\times n}$ from a uniform distribution on $\{-l, -l+1, \ldots, l-1, l\}$, where according to the authors, the value of the bound l has negligible impact on the quality of the generated basis (the authors chose ± 4 in their implementation). A second method is to first generate a square lattice $L(kI)$ for some positive integer k, then add a small amount of noise $R' \in \{-l, \ldots, l\}^{n\times n}$. With this method of sampling the private basis, it is important to balance the choice of values between k and l, where a larger k value gives a more orthogonal basis, but also weakens the security of the trapdoor function by making it easier to reduce the public basis into a short, orthogonal basis using basis reduction algorithm.

The authors also described two methods for generating the public basis B from the private basis. The first method is to directly generate random unimodular matrix T and set $B = TR$, then repeat until satisfactory. A second method is to repeatedly apply column mixing, where at each mixing a column of R is chosen, and a linear combination of all other columns is added to the chosen column.

Mathematically, the two methods are equivalent. However, in implemenetation, we would like the values of B to be smaller for space efficiency while maintaining sufficient security so that the function cannot be easily inverted using the public basis alone and so that B cannot be easily reduced using basis reduction algorithm. The authors preferred column mixing for requiring less computation and for producing public basis B with smaller values.

Unfortunately, there is no known rigorous description of how skewed B needs to be for the trapdoor function to be secure. The authors relied on experimental methods and determined that for $n \approx 100$, $2n$ steps of column mixing is enough to render LLL basis reduction ineffective at meaningfully improving the quality of the public basis B.

4.4 Inverting the function without the trapdoor

From a high level, inverting the function corresponds to finding the an approximate closest lattice point, so the best attacks on the trapdoor are also algorithms for solving the CVP problem.

The most straightforward attack on the trapdoor is to simply run the inversion algorithm with the public basis instead of the private basis:

$$B^{-1}\mathbf{x} = B^{-1}(B\mathbf{v} + \mathbf{e}) = \mathbf{v} + B^{-1}\mathbf{e}$$

Due to B being very skewed, this procedure does not immediately yield the correct value for $\mathbf{v}$ since $B^{-1}\mathbf{e}$ is not $\mathbf{0}$. However, since the possible values of $\mathbf{e}$ is finite, we can perform an exhaustive search on all possible values of $\mathbf{d} = B^{-1}\mathbf{e}$, although the search space will grow exponentially with the number of dimensions n. In experimental settings, with $n \geq 100$ the search space reaches 168 bits of entropy.

Using a better approximation algorithm for CVP, such as the nearest plane algorithm, yields a more efficient inversion than the brute-force search described above, although the complexity of the algorithm still scales exponentially with n. In experimental settings, with $n = 150$ the workload of the nearest plane algorithm reaches 104 bits.

4.5 Public key cryptosystem

The authors of the GGH paper presented a public key cryptosystem that makes direct usage of the trapdoor. The main security parameter is n the number of dimension of the lattice. The secret key is the short, orthogonal basis R generated by applying a small amount of noise to a truly orthogonal basis. The public key is the long basis B generated by applying column mixing to R. The encryption function is exactly the evaluation of the trapdoor, and the decryption function is the inversion of the trapdoor.

What remains is the question of ”where to encode the message”, for which the authors discussed a few options:

1. Use a generic encoding that takes advantage of the hard-core bits of the one-way function, although this encoding is inefficient (it only encodes $\log n$ bits at a time), and it does not take advantage of any specific features of the trapdoor construction itself

2. Directly encode the message into a lattice point $\mathbf{v}$. However, this scheme is insecure because an adversary can compute $B^{-1}\mathbf{c} = \mathbf{v} + B^{-1}\mathbf{e}$, where $B^{-1}\mathbf{e}$ might not be big enough to obscure all information about $\mathbf{v}$, which means this adversary can obtain partial information about the message

3. Encode the message into the least significant bits of each entry of $\mathbf{v}$ and choose all other bits randomly. The authors argued that with appropriate choice of probability distribution for the error term $\mathbf{e}$, no polynomial-time adversaries will be able to distinguish the parity of the entries of the ciphertext from truly random, thus achieving IND-CPA security.

Later Micciancio also proposed to encode the message in the error term $\mathbf{e}$, and instead chooses the lattice point $\mathbf{v}$ at random. However, this scheme is not trivial because special care is needed for security requirement, and we also have the problem of a lattice being a countably infinite set, making defining probability distribution a non-trivial problem.

4.6 Digital signature

The GGH trapdoor is also suitable for a digital signature: the message is an arbitrary point in the linear span of the good basis, and the signature is the (approximate) closest lattice point, which can be efficiently computed using the good basis R. The verifier checks that the signature is valid by first using the public basis to verify that the signature is a lattice point, then computing the norm between the signature and the message to check that the distance is sufficiently small.

5 Conclusion and final comments

In this project, we reviewed some fundamental results of lattices, analyzed the difficulty of hard lattice problems and the efficiency of their best known solutions. We then discussed a lattice trapdoor construction that takes advantage of an exponential gap between short basis and long basis in their ability to solve the closest vector problem, and applied the trapdoor to build a public key cryptosystem and a digital signature scheme.

Both the public-key cryptosystem and the signature scheme have been cryptanalyzed (by Oded Regev and Phong Q. Nguyen respectively) in their original proposed form. However, the idea of using a pair of short and long basis to build trapdoors has been extended in later studies. Gentry, Peikert, and Vaikuntanathan developed this idea into the GPV framework that later led to FALCON, a digital signature scheme that was submitted to NIST's post-quantum cryptography competition. This idea was also behind NTRUSign, whose post-quantum variant was also submitted to NIST's post-quantum cryptography competition.