A Review of Cryptographic Hardness for Learning Intersections of Halfspaces (by Adam Klivans & Alexander Sherstov)

Ahmad Chatha (ac3877), Preeti Vaidya (pv2259)

September 18, 2018

Abstract. This paper demonstrates that the problem of PAC learning intersections of halfspaces is at least as hard as solving lattice based cryptography problems. Specifically, the authors prove that if we have a polynomial time PAC (improper) algorithm for learning intersection of n^ϵ halfspaces, then we can solve the unique shortest vector problem (uSVP). They also prove that having a PAC algorithm for intersections of n^ϵ low-weight halfspaces gives a polynomial-time quantum algorithm which solves the shortest vector problem (SVP) and shortest independent vector problem (SIVP). They also extend their results to depth-2 neural networks and depth-3 arithmetic circuits. Here we only focus on the main results of the paper.

1 Introduction

Learning a single halfspaces (linear threshold function) is an extremely well studied problem in many different areas of computer science. A natural extension of that problem is learning the intersection of k halfspaces.

1.1 Intersections of linear threshold functions

Let h be a hyperplane in $\mathbb{R}^n$ such that $h = \{x : \sum_{i=1}^n w_i x_i = \theta\}$. This hyperplane induces a boolean function, described by, $f(x) = sign(\sum_{i=1}^n w_i x_i - \theta)$ which describes the halfspace or the linear threshold function.

Learning the intersections of k halfspace, is still a challenging problem. This is an important problem as there are several examples of applications of this type of a problem: for example, any convex body can be expressed as an intersection of halfspaces. In order to understand the delimiters in the approach to solving the problem, let us start first, by getting a formal understanding of the problem of identifying the intersection of half spaces, as stated by Khot and Saket [2],

Definition 1. An instance of INTERSECTION-HALFSPACE is a set of points in $\mathbb{R}^n$ each labeled either '+' or '-' and the goal is to find a function of at most k linear threshold functions (halfspaces) which correctly classifies the maximum number of points, where a '+' point is classified correctly if it lies inside the intersection and a '-' point is classified correctly if it lies outside of it.

The problem of learning a single halfspace is well understood. We have efficient algorithms (linear programming, perceptron) which can learn a single halfspace and we know a good deal about the performance guarantees of such algorithms. Naively, one might think that since we can learn a single halfspace efficiently and we can learn conjunctions efficiently, the problem of learning AND of several halfspaces should be easy (learn individual halfspaces and AND them together). However, the issue is that the examples given to us by the oracle are labeled according to the intersections of several halfspaces and not a single one. So we cannot ever predict which halfspaces were responsible for labeling a given point and as a result, the ability to learn a single halfspace doesn't help in any way.

Moreover, it has been proven that the problem of learning intersection of even two halfspaces is hard. Blum and Rivest [5] showed that it is NP-hard to learn the intersection of two halfspaces with intersection of two halfspaces, and Alekhnovich, Braverman, Feldman, Klivans and Pitassi [6] proved a similar result when the hypothesis is an intersection of constant number of halfspaces.

1.2 Lattice based Cryptography

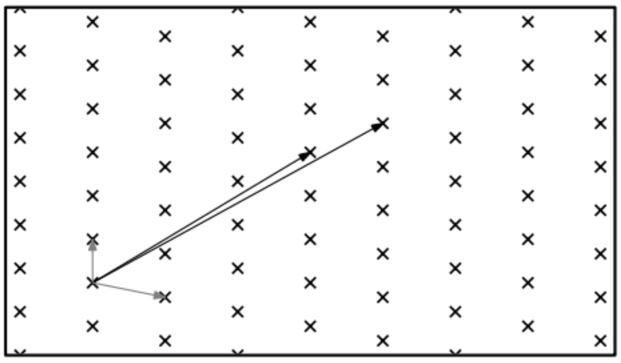

Figure 1: A lattice in $\mathbb{R}^2$ and two of its basis. [7]

A *lattice* is the set of all integer combinations of n linearly independent *basis* vectors $v_1, ..., v_n$ in $\mathbb{R}^n$. In other words, its a discrete additive subgroup of R^n. Lattice based cryptosystems basically try to hide the structure of the lattice. For the purpose of this paper, and generally too, the major problems on lattices are pertaining to the shortest vector problem. (See Section 1.3).

This is an important concept as there are several examples of applications of this type of a problem: such as integer programming, relation finding for integers, integer factoring, and Diophantine approximation (deals with the approximation of real numbers using rational numbers).

The starting point of this discussion on Lattice Based Cryptography is Ajtai's [8] discovery that lattices, can actually be used to construct cryptographic primitives and the security premise was dependent on the worst-case hardness of lattice problems. In contrast, average-case assumptions are leveraged in almost all other cryptographic constructions.

Another reason for exploring lattice based cryptography is that the computations involved often require modular operations and this can be leveraged in scenarios when encryption is performed by a low-cost device. Lattice based cryptography is an important resource as currently there aren't many alternatives to traditional number-theoretic based cryptography such as RSA. Moreover, no quantum algorithm is yet known to break a lattice based cryptosystem.

The cryptosystem described below is the general model that we will follow for the following discussion and results. The basic encryption is randomized and is done on one-bit messages (0 and 1). The decryption process is deterministic. Let us define, first, the basic definition that we are going to follow,

Definition 2. $e_{K,r} : \{0,1\} \to \{0,1\}^{poly(n)}$ *denotes the encryption function corresponding to a choice of private and public keys* $K = (K_{priv}, K_{pub})$ *and a random string* r*.*

Using the above basic definition of the cryptosystem's encryption function, let us proceed to understanding the basic ideology the system will follow in the scenario when it is presented with an encryption of 0 and 1. This notion is represented below,

Definition 3: (Distinguisher) An algorithm A is said to distinguish between the encryptions of 0 and 1 if for some universal constant c,

$$|Pr_{K,r}[A(K_{pub}, e_{K,r}(1)) = 1] - Pr_{K,r}[A(K_{pub}, e_{K,r}(0)) = 1] \geqslant \frac{1}{n^c}$$

1.3 Lattice Problems

The shortest vector problem (SVP) is one of the main computation problems that are associated with lattices. In SVP, the goal is to return a non-zero vector in the lattice such that the norm of this vector is the shortest nonzero lattice vector by at most some approximation factor γ, given the description of the lattice basis.

For the purpose of this discussion, we focus on three kinds of lattice problems:

1. *f(n)-SVP*: The goal is to approximate the length of a shortest nonzero vector within a factor of f(n).
2. *f(n)-uSVP*: This is an extension of the first problem and the goal is to find a shortest nonzero vector in the lattice, provided that it is shorter by a factor of at least f(n) than any other non-parallel vector.
3. *f(n)-SIVP*: The goal is to output a set of n linearly independent lattice vectors of length at most $f(n) * opt$, where opt is the minimum length over all sets of n linearly independent vectors from the lattice (the length of a set is the length of its longest vector).

The important fact to consider in the above description of the problems associated with lattices is that the hardness of these problems depends in the choice of *f(n)*, such that for certain smaller choices of *f(n)*, these problems become NP-Hard. All three problems become harder as the approximation factor $1 \leq f(n) \leq poly(n)$ decreases. For the purpose of this discussion, the author chose $f(n) = O(n^{1.5})$, an approximation factor for which these three problems are believed to be hard.

2 Previous Work

In this section, we point out the previous work discussed in the original paper and as discussed in any other materials referred by us.

Let us first evaluate the previous work done in the field of learning intersection of halfspaces. For learning intersection of halfspaces, there are various special case algorithms. When the data points are drawn from the uniform distribution over the unit ball, Blum, Kannan [9] and Vempala [10] gave algorithms to PAC-learn intersection of a constant number of halfspaces. For the uniform distribution over the boolean hypercube, Klivans, O'Donnell and Servedio [11] obtained an algorithm for learning intersection of constant number of halfspaces. Arriaga and Vempala [12] and Klivans and Servedio [13] gave algorithms for learning intersection of halfspaces when no data point is too close to any separating hyperplane. Despite all these specialized case discussion on solving the problem, the generic learning intersection of halfspaces is still a problem.

Let us now evaluate some of the previous work done on lattice problems. Currently, as discussed by Regev [14], lattice problems, such as the shortest vector problem (SVP) and the closest vector problem (CVP), from a computational complexity point of view are consolidated in the figure below.

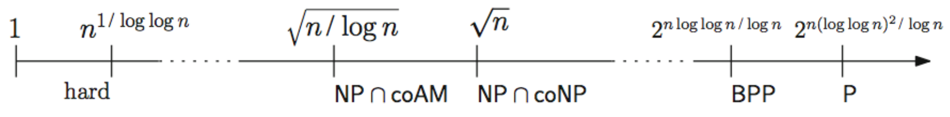

Figure 2: The complexity of lattice problems (some constants omitted) [14]

Additionally, on public key cryptosystem based on a worst-case lattice problem, Ajtai and Dwork [15] presented a public key cryptosystem based on the worst-case hardness of $O(n^8)$-uSVP. Then, in [16], Goldreich, Goldwasser and Halevi showed how to eliminate decryption errors that existed in the original scheme. They also improved the security to $O(n^7)$-uSVP. In this paper, we are studying the evaluation with $f(n) = O(n^{1.5})$ lattice problems.

3 Connecting Cryptography & Learning

In this section, we discuss the relationship between the security of a public-key cryptosystem and the hardness of learning an associated concept class, established by Kearns and Valiant [17].

The author re-derives this result with an extension to allow for errors in the decryption process. The idea follows from the thought that given a large pool of encryptions, we can use these as a training set to learn the decryption function. This, however, is undesirable naturally due to the security concerns regarding the cryptosystem, defeating the very purpose. We assume that the cryptosystem is secure and thus, conclude that the decryption function cannot be learnt. This is described in the lemma below:

Lemma 3.1 Cryptography and learning; (cf. Kearns Valiant [17])

Consider a public-key cryptosystem for encrypting individual bits by n-bit strings. Let C be a concept class that contains all the decryption functions $d_K : \{0,1\}^n \rightarrow \{0,1\}$ of the cryptosystem, one for each choice of key $K = (K_{priv}, K_{pub})$. Let $\epsilon(n) = Pr_{K,r}[d_K(e_K, r(0)) \neq 0$ or $d_K(e_K, r(1)) \neq 1]$ be the probability of decryption error (over the choice of keys and randomization in the encryption). If C is weakly PAC-learnable in time t(n) with $t(n)\epsilon(n) = \frac{1}{n^{\omega(1)}}$, then there is a distinguisher between the encryptions of 0 and 1 that runs in time O(t(n)).

Proof:
From the definition above, let $e_{K,r} : \{0,1\} \rightarrow \{0,1\}^n$ denote the encryption function corresponding to a choice of private and public keys K = (K_{priv}, K_{pub}) and a random string r. Subsequently, let $d_K : \{0,1\}^n \rightarrow \{0,1\}$, denote the respective decoding function d_K for the given encryption function. Further, assuming that C is learn-able with respect to the O(t(n)) *Distinguisher* (See Definition above) A. From the definition of A, we have for some universal constant c, as long as $t(n)\epsilon(n) = \frac{1}{n^{\omega(1)}}$,

$|Pr_{K,r}[A(K_{pub}, e_{K,r}(1)) = 1] - Pr_{K,r}[A(K_{pub}, e_{K,r}(0)) = 1]| \geqslant \frac{1}{n^c}$
From the above equation and discussion, we can clearly see that the equation and the probabilities, themselves, are dependent on the choice of keys, K_{pub}, K_{priv}, the randomness of the encryption function, $e_{K,r}$, and the internal randomization offered by A. Thus, we can say that A is the desired *distinguisher*.

Let us now understand the working of the *distinguisher* Algorithm A. A takes as input a pair (K_{pub}, w), where $w \in 0, 1^n$ is the encryption of an unknown bit. The first step in the process is that A draws t(n) independent training examples, choosing each as described below:

1. Pick $b = 0$ or $b = 1$, with equal probability.
2. Pick r, an unbiased random string.
3. Create a training example $\langle e_{K,r}(b), b \rangle$.

The next step for A is to use these training samples to learn the hypothesis h for learning C. Assume that there is an algorithm for learning C. We also assume that d_K is consistent with the samples and no decryption error has occurred. Thus, we get an hypothesis that is consistent with the constraint below,

$$Pr_{b,r}[he_{K,r}(b)) = d_K(e_{K,r}(b))] \geqslant \frac{1}{n^c} + \frac{1}{2}$$

For the purpose of this discussion, we have assumed that the learner is guaranteed to find the hypothesis h that satisfies the above constraint, given consistent training examples.

The goal of A is to output *h(w)* and exit. However, let us first evaluate whether A is a *distinguisher*, in order to obtain some useful intermediate results. This can be done in the following cases:

No Decryption Error: In the event, $\bar{\varepsilon}$, when no decryption error occurs, we have a bound as:

$$Pr_{K,r}[A(K_{pub}, e_{K,r}(1)) = 1|\bar{\varepsilon}] - Pr_{K,r}[A(K_{pub}, e_{K,r}(0)) = 1|\bar{\varepsilon}] \geqslant \frac{2}{n^c} - 2\epsilon(n)$$

.

Decryption Error: In the event, ε, the likelihood of a decryption error is small (by union bound) on any run of algorithm A, we have a bound as:

$$Pr[\varepsilon] \leq t(n)\epsilon(n)$$

.

Using the above results, we have,

$Pr_{K,r}[A(K_{pub}, e_{K,r}(1)) = 1] - Pr_{K,r}[A(K_{pub}, e_{K,r}(0)) = 1] \geqslant$

.

$(Pr_{K,r}[A(K_{pub}, e_{K,r}(1)) = 1|\bar{\varepsilon}] - Pr_{K,r}[A(K_{pub}, e_{K,r}(0)) = 1|\bar{\varepsilon}]) - 2Pr[\varepsilon]$

Substituting the above results, we can say that,

$$Pr_{K,r}[A(K_{pub}, e_{K,r}(1)) = 1] - Pr_{K,r}[A(K_{pub}, e_{K,r}(0)) = 1] \geqslant \frac{1}{n^c}$$

Thus, we successfully proved that A is the *distinguisher* between the encryptions of 0 and 1.

4 Using intersections of degree-2 PTFs to learn decryption functions

Lemma 3.1 showed that breaking a cryptosystem amounts to learning its decryption function with in the PAC settings. In other words, if we have an efficient PAC algorithm for computing the decryption function of a cryptosystem, then the cryptosystem is insecure. We will use intersections of degree-2 polynomial threshold functions to learn the decryption functions since the decryption functions contain PARITY as a subfunction which cannot be learned by intersections of halfspaces. So we will have to prove the problem of learning intersections of halfspaces is equivalent to the problem of learning intersections of degree-2 PTFs.

Lemma 4.1 *If there is a (weak) PAC algorithm for learning intersections of n^ϵ light halfspaces, then there is a (weak) PAC algorithm for learning intersections of n^c light degree-2 PTFs.*

Proof. The idea is to define three concept classes and establish that learning in one implies learning in the other. Let $\mathcal{C}$: intersections of n^ϵ light halfspaces, $\mathcal{C}'$: intersections of n^ϵ light degree-2 PTFs and $\mathcal{C}''$: intersections of n^c light degree-2

PTFs.

Note that a degree-2 PTF in n variables $x_1, x_2, ..., x_n$ can be expressed as a linear threshold function in $n + \binom{n}{2}$ variables $x_1, ..., x_n, x_1x_2, x_1x_3, ..., x_{n-1}x_n$. So given a (weak) PAC algorithm A for $\mathcal{C}$,$(\epsilon, \delta) > 0$ and an oracle access to $EX(c, \mathcal{D})$ where $\mathcal{D}$ is the distribution over $x_1, x_2, ..., x_n$, we can PAC learn $\mathcal{C}'$. We construct a new non-uniform distribution $\mathcal{D}'$ over $n + \binom{n}{2}$ variables by the polynomial-time map from $x_1, x_2, ..., x_n \Rightarrow x_1, ..., x_n, x_1x_2, x_1x_3, ..., x_{n-1}x_n$. We run A on $EX(c, \mathcal{D}')$ and obtain a ϵ-accurate hypothesis h for $\mathcal{C}'$ with probability $\geq 1 - \delta$.

We also note that $\mathcal{C}' \subseteq \mathcal{C}''$. Using the standard padding argument, we can turn a PAC learning algorithm A for $\mathcal{C}'$ into one that learns $\mathcal{C}''$ for any constant $c > 0$ by padding the input given to A. The new algorithm basically ignores the padded zeros and run A on the padded input to learn the bigger concept class. Therefore, given a PAC algorithm for learning $\mathcal{C}$ gives a PAC algorithm for $\mathcal{C}'$ and subsequently for $\mathcal{C}''$. $\square$

4.1 Learning the Decryption function of uSVP-based Cryptosystem as an intersections degree-2 PTFs

The cryptosystem based on the unique shortest vector problem is described below:
Private Key: A real number H such that $\sqrt{N} \leq H < 2\sqrt{N}$.
Public Key: A vector $(A - 1, ...A - m, i_0)$, where $i_0 \in \{1, ..., m\}$ and each $A_i in\{0, ..., N - 1\}$.
Encryption: To encrypt 0, we pick a random $S \subseteq [m]$ and output $\sum_{i \in S} A_i \mod N$. To encrypt 1, we again pick a random S and output $\lfloor A_{i_0}/2 \rfloor + \sum_{i \in S} A_i \mod N$.
Decryption: Upon getting $W \in \{0, ..., N-1\}$, we decrypt to 0 if frac$(WH/N) < 1/4$, and 1 otherwise. By frac(a), we mean the distance from $a \in \mathbb{R}$ to the closest integer. We can let $A = H/N$ then the decryption function is about finding out if $frac(AW) < 1/4$ or not(representing H/N to with poly(n) fractional bits does not affect the decryption function).

Regev [18] proved the following theorem about solving uSVP problem. We will use it in the main results.
Theorem 2.2(Regev [18]) Assume that there is a polynomial-time distinguisher between the encryptions of 0 and 1. Then there is a polynomial-time solution to every instance of $(\sqrt{n} \cdot \gamma(n))$-uSVP.

The basic idea behind the decryption function of uSVP is to figure out if the encryption is close to a multiple of $A = H/N$ or not. If it is within $\pm 1/4$ of A then we output 0, otherwise 1. A natural Boolean predicate that encapsulates this idea is NEAR-INT(AW) $= 1 \iff$ frac$(a) < 1/4$. In other words, if the predicate outputs 1 then that means we decrypt to 0, and if the predicate outputs 0, then we decrypt to 1. Note that there are infinite multiples of H/N and we only care about if the encryption is close to (within a factor of 1/4) any one of them. In other words, we only care about the fractional part of AW and ignore the integral part. With the predicate in mind, we state the lemma:

Lemma 4.2 *Let $A > 0$ be a real number with k fractional bits. Then the function $f(x) = NEAR\text{-}INT(A\sum_{j=0}^{n-1} x_j 2^j)$ can be computed by the intersection of k degree-2 PTFs.*

Proof. The high level idea is to spread out the decryption function $f(x) =$ NEAR-INT$(A\sum_{j=0}^{n-1} x_j 2^j)$ over small intervals which can be learned by an AND of k degree-2 PTFs.
As mentioned before, we only concern ourselves with the fractional part of A. Let $\{A\} = b_1, ...b_k$ be the fractional part of A where each $b_i \in \{0, 1\}$. Let $S(x) = \{A\sum_{j=0}^{n-1} x_j 2^j\}$. Then using the fractional part of A and ignoring the whole number terms (1.000..,2.000..) we can write the fractional part of $S(x)$ as

$$\{S(x)\} = \{A\sum_{j=0}^{n-1} x_j 2^j\} = \{\sum_{i=1}^{k}\sum_{j=0}^{n-1} b_i x_j 2^{j-1}\} = \{\sum_{i=1}^{k}\sum_{j=0}^{min\{n-1,i-1\}} b_i x_j 2^{j-1}\}$$

Note that $S(x)$ is a multiple of $1/2^k$ and its range is $[0, k]$. We divide the range into small intervals such that on a particular interval, the value of NEAR-INT$(S(x))$ is constant (0 or 1). The following figure shows the intervals and the values of $S(x)$ and NEAR-INT$(S(x))$.

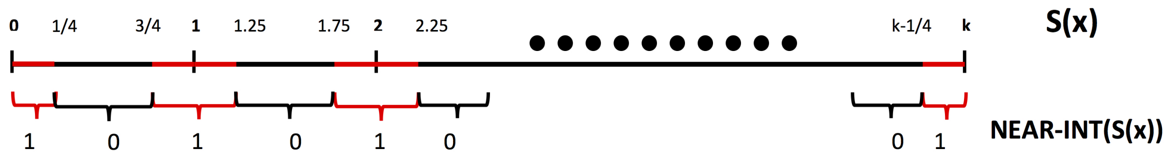

Figure 3: Value of the decryption function is constant on small intervals.

Each interval with its associated NEAR-INT$(S(x))$ value can be recognized as the following polynomial threshold function:

$$(S(x) - \frac{a+b}{2})^2 \leq (\frac{b-a}{2})^2$$

For example, take $S(x)$ to be in the third interval, then $(S(x) - 1)^2 \leq \frac{1}{16}$ for $3/4 > S(x) < 5/4$ which means that the predicate is true (NEAR-INT$(S(x))$=1) and we decrypt to 0. Think of the interval $[0, k]$ as chopped into little red and black pieces where the $\pm\frac{1}{4}$ region around all the integers have the predicate set to true (red) and for other regions, the predicate outputs false (black). The security of the cryptosystem relies on the fact that any attacker cannot predict with a non-negligible probability that the encryption will fall into the red region or the black region.
To align the output of the decryption function with the PTF, we negate the interval by replacing the $\leq$ inequality sign by $>$. This way, when the PTF outputs 1, then the decryption function also outputs 1 (NEAR-INT$(S(x))$=0) and we obtain a natural mapping between the two. Also, it is clear from the figure above that after the first interval, every two consecutive intervals (black & red) cover a distance of 1, so to cover all of $[0, k]$ we need at most $2k + 1$

intervals. So if we AND the negation of all k intervals on which the PTF outputs $1 \iff$ decryption outputs $1 \iff$ NEAR-INT($S(x)$)=0, we get the final function as intersections of k degree-2 polynomial threshold functions, which is what we wanted. □

4.1 Learning the Decryption function of SVP- and SIVP-based Cryptosystem as an intersections degree-2 PTFs

The cryptosystem based on SVP and SIVP is similar to the uSVP described before, except here we deal with vectors instead of real numbers. It is described below: **Private Key:** A vector s in $\mathbb{Z}_p^N$.
Public Key: A sequence of pairs $(a_1, b_1), ..., (a_m, b_m)$ where each $a_i \in \mathbb{Z}_p^N$ and each $b_i \in \mathbb{Z}_p$.
Encryption: To encrypt 0, we pick a random $S \subseteq [m]$ and output $(\sum_{i \in S} a_i, \sum_{i \in S} b_i)$. To encrypt 1, we again pick a random S and output $(\sum_{i \in S} a_i, \lfloor p/2 \rfloor + \sum_{i \in S} b_i)$.
Decryption: Upon getting $(a, b) \in \mathbb{Z}_p^N \times \mathbb{Z}_p$, we decrypt to 0 if $b - < a, s >$ is closer to 0 than to $\lfloor p/2 \rfloor$ modulo p, and 1 otherwise.

Regev [19] proved the following theorem about solving SVP and SIVP problems. We will use it in the main results.
Theorem 2.3(Regev [19]) Assume that there is a polynomial-time (possibly quantum) algorithm for distinguishing between the encryptions of 0 and 1. Then there is a polynomial-time quantum solution to SVP and SIVP.

The idea behind the decryption function of SVP and SIVP based cryptosystem is to figure out if the encryption contains an error term or if it was chosen uniformly randomly. Just like in the case of uSVP cryptosystem, we again describe a natural Boolean predicate which encapsulate the decryption function:

$$\text{NEAR-MID}_p(a) \iff |b - \text{Round}(p/2)| \leq \min\{b, p - b\}$$

where Round(c) is the nearest integer function and $b \in \{0, 1, ..., p-1\}$ and a are congruent modulo p. Then $d_{s1,...,sn}(b, a_1, ..., a_n) = \text{NEAR-MID}_p(b - \sum a_i s_i)$ is exactly computes the decryption function of SVP and SIVP based cryptosystem.

Lemma 4.3 *Let* $d_{s1,...,sn} : (\{0,1\}^{\log p})^{n+1} \Rightarrow \{0,1\}$ *be the Boolean function defined by*

$$d_{s1,...,sn}(x) = \textit{NEAR-MID}_p(\sum_{i=0}^{\log p - 1} 2^i x_{0,i} - \sum_{j=1}^{n} s_j \sum_{i=0}^{\log p - 1} 2^i x_{j,i})$$

where all s_i are integers in $\{0, ..., p-1\}$. Then $d_{s1,...,sn}$ can be computed by the intersection of $n \log p$ degree-2 PTFs.

Proof. The proof is similar to the proof of previous lemma. Again, the idea is to divide the range of the decryption function $d_{s1,...,sn}$ into intervals which can be learned by an AND of some number of degree-2 PTFs.
So let

$$S(x) = \sum_{i=0}^{\log p - 1} 2^i x_{0,i} - \sum_{j=1}^{n} s_j \sum_{i=0}^{\log p - 1} 2^i x_{j,i} = \sum_{i=0}^{\log p - 1} 2^i x_{0,i} - \sum_{j=1}^{n} \sum_{i=0}^{\log p - 1} (2^i s_j \text{mod p}) x_{j,i}$$

. Then the decryption function is equal to NEAR-MID$_p(S(x))$. Note that $S(x)$ is an integer in the range of $-(p-1)n\log p$ and $p-1$. Just like the previous lemma, we divide the length of range of $S(x) < pn\log p$ into consecutive intervals on which NEAR-MID$_p(S(x))$ is constant. Every two consecutive intervals, on which the NEAR-MID$_p(S(x))$ changes values, cover a length of p. So the total number of consecutive intervals are $2(pn\log p)/p = 2n\log p$. In order to map the output of the decryption function to the output of the intersection of PTFs, we negate the intervals on which NEAR-MID$_p(S(x)) = 0$. There are $n\log p$ such intervals and by taking AND of their negations, we get the intersection of $n\log p$ degree-2 PTFs we wanted. □

5 Main Results

At this point, we have all the ingredients we need to prove the main results of the paper.

Theorem 1.1 Assume that we have a PAC algorithm for learning intersections of n^ϵ halfspaces in n dimensions, then there is a polynomial time solution to $\tilde{O}(n^{1.5})$-uSVP.

Proof. Let $\mathcal{C}$ denote the concept class of intersections of n^ϵ halfspaces and $\mathcal{C}'$ be the concept class of intersections of n^c degree-2 polynomial threshold functions. By lemma 4.1, if there is a PAC algorithm for $\mathcal{C}$ then there is a PAC algorithm for $\mathcal{C}'$. That means we can rely on the PAC algorithm for $\mathcal{C}'$ to learn the decryption function. By lemma 4.2, the decryption functions of uSVP based cryptosystem are in $\mathcal{C}'$ and we have a PAC algorithm for learning $\mathcal{C}'$ using intersection of k degree-2 PTFs that runs in polynomial time. By lemma 3.1, this PAC algorithm gives us a distinguisher which can tell the difference between the encryption of 0 and 1 with non-negligible probability. Finally, using Theorem 2.2, since we have a polynomial time distinguisher, we have a polynomial solution to every instance of uSVP problem. □

Theorem 1.2 Assume that we have a PAC algorithm for learning intersections of n^ϵ halfspaces in n dimensions (for $\epsilon > 0$), then there is a polynomial time solution to $\tilde{O}(n^{1.5})$-SVP and $\tilde{O}(n^{1.5})$-SIVP.

Proof. The proof is almost exactly like the proof above. We let $\mathcal{C}$ and $\mathcal{C}'$ be the concept classes as described above. By lemma 4.1, PAC algorithm for $\mathcal{C} \Rightarrow$ PAC algorithm for $\mathcal{C}'$. By lemma 4.3, the decryption functions of SVP and SIVP based cryptosystem are in $\mathcal{C}'$ *(here the authors wrote uSVP by mistake)* and we have a PAC algorithm for learning $\mathcal{C}'$ using intersection of k degree-2 PTFs that runs in polynomial time. By lemma 3.1, having a PAC algorithm gives us a distinguisher between the encryption of 0 and 1 that runs in polynomial time. Using Theorem 2.3, since we have a polynomial time distinguisher, that means we have a efficient quantum solution to every instance of SVP and SIVP problem. □

6 Conclusion

Hardness results are mostly proved using reduction techniques where we transform one problem into another. In this paper, we reduced the problem of learning the decryption functions of uSVP,SVP,SIVP-based cryptosystems to the problem of learning intersection of n^c degree-2 polynomial threshold functions. We further reduced the problem of learning intersection of n^c degree-2 PTFs to learning intersection of n^ϵ halfspaces. We also proved that if we can learn the decryption function efficiently then we can distinguish between the encryptions of 0 and 1 and hence, break the cryptosystem. By combining all these results, we were able to show that the problem of learning intersection of halfspaces is at least as hard as breaking SVP based cryptosystems (and its variants). In other words, if someone comes up with an efficient algorithm for learning intersections of halfspaces, then they will be able to break several lattice-based cryptosystems. However, because these cryptosystems are based on lattice problems that are proven to be hard, it is very unlikely that someone will be able to learn intersections of halfspaces efficiently.

References

1 Klivans, O'Donell and Servedio. Learning intersections and thresholds of half spaces.

2 Khot and Saket. On Hardness of Learning Intersection of Two Halfspaces.

3 Feldman, Gopalan, Khot, and Ponnuswami. New results for learning noisy parities and halfspaces.

4 Guruswami and Raghavendra. Hardness of learning halfspaces with noise.

5 Blum and Rivest. Training a 3-node neural network is NP-complete.

6 Alekhnovich, Braverman, Feldman, Klivans, and Pitassi. Learnability and automatizability.

7 Oded Regev. Lattice-based Cryptography.

8 Ajtai. Generating hard instances of lattice problems.

9 Blum and Kannan. Learning an intersection of a constant number of halfspaces over a uniform distribution.

10 S. Vempala. A random sampling based algorithm for learning the intersection of half-spaces.

11 A. Klivans, R. O'Donnell, and R. Servedio. Learning intersections and thresholds of halfspaces.

12 R. Arriaga and S. Vempala. An algorithmic theory of learning: Robust concepts and random projection.

13 A. Klivans and R. Servedio. Learning intersections of halfspaces with a margin.

14 Oded Regev. On the Complexity of Lattice Problems with Polynomial Approximation Factors.

15 M. Ajtai and C. Dwork. A public-key cryptosystem with worst-case/average-case equivalence.

16 O. Goldreich, S. Goldwasser, and S. Halevi. Eliminating decryption errors in the Ajtai-Dwork cryptosystem.

17 M. Kearns, L. Valiant, Cryptographic limitations on learning Boolean formulae and finite automata.

18 Oded Regev. New lattice based cryptographic constructions.

19 Oded Regev. On lattices, learning with errors, random linear codes, and cryptography.