Proceedings of Machine Learning Research vol 99:1–35, 2019 32nd Annual Conference on Learning Theory

# Computational Limitations in Robust Classification and Win-Win Results

**Akshay Degwekar** AKSHAYD@MIT.EDU
*Massachusetts Institute of Technology*

**Preetum Nakkiran** PREETUM@CS.HARVARD.EDU
*Harvard University*

**Vinod Vaikuntanathan** VINODV@MIT.EDU
*Massachusetts Institute of Technology*

**Editors:** Alina Beygelzimer and Daniel Hsu

## Abstract

We continue the study of statistical/computational tradeoffs in learning robust classifiers, following the recent work of Bubeck, Lee, Price and Razenshteyn who showed examples of classification tasks where (a) an efficient robust classifier exists, *in the small-perturbation regime*; (b) a non-robust classifier can be learned efficiently; but (c) it is computationally hard to learn a robust classifier, assuming the hardness of factoring large numbers. Indeed, the question of whether a robust classifier for their task exists in the large perturbation regime seems related to important open questions in computational number theory.

In this work, we extend their work in three directions.

First, we demonstrate classification tasks where computationally efficient robust classification is impossible, even when computationally unbounded robust classifiers exist. For this, we rely on the existence of average-case hard functions, requiring no cryptographic assumptions.

Second, we show hard-to-robustly-learn classification tasks *in the large-perturbation regime*. Namely, we show that even though an efficient classifier that is *very robust* (namely, tolerant to large perturbations) exists, it is computationally hard to learn any non-trivial robust classifier. Our first construction relies on the existence of one-way functions, a minimal assumption in cryptography, and the second on the hardness of the learning parity with noise problem. In the latter setting, not only does a non-robust classifier exist, but also an efficient algorithm that generates fresh new labeled samples given access to polynomially many training examples (termed as generation by Kearns et. al. (1994)).

Third, we show that any such counterexample implies the existence of cryptographic primitives such as one-way functions or even forms of public-key encryption. This leads us to a win-win scenario: either we can quickly learn an efficient robust classifier, or we can construct new instances of popular and useful cryptographic primitives.

## 1. Introduction

The basic task in learning theory is to learn a classifier given a dataset. Namely, given a labeled dataset $\{(X_i, f(X_i))\}_{i\in[n]}$ where $f$ is the unknown ground-truth and $X_i$ are drawn i.i.d. from a distribution $D$, learn a classifier $h$ so as to (approximately) minimize

$$\delta := \mathop{\mathbb{P}}_{X\sim D}[h(X) \neq f(X)]$$

Adversarial machine learning is harder in that the learned classifier is required to be *robust*. Namely, it has to produce the right answer even under bounded perturbations (under some distance measure) of the sample $X \sim D$. That is, the goal is to learn a classifier $h$ so as to (approximately) minimize

$$\overline{\delta} := \mathop{\mathbb{P}}_{X \sim D}[\exists Y \in B(X, \varepsilon) \text{ s.t. } h(Y) \neq f(Y)]$$

where $B(X, \varepsilon) = \{Y : d(X, Y) \leq \epsilon\}$ and $d$ is the distance measure in question.

Learning *robust* classifiers is an important question given a large number of attacks against practical machine learning systems that show how to minimally perturb a sample $X$ so that classifiers output the wrong prediction with high probability. Such attacks were first discovered in the context of spam filtering and malware classification Dalvi et al. (2004); Lowd and Meek (2005); Biggio and Roli (2018) and more recently, following Goodfellow et al.; Szegedy et al. (2013), in image classification, voice recognition and many other domains.

This state of affairs raises a slew of questions in learning theory. Fix a concept class $\mathcal{F}$ and a distribution $D$ for which efficient (non-robust) learning is possible. Do there exist robust classifiers for $F$? Do there exist *efficiently computable* robust classifiers for $F$? Pushing the envelope further, can such classifiers be learned with small sample-complexity? and finally, is the learning algorithm computationally efficient? The answer to these questions give rise to five possible worlds of robust learning, first postulated in two recent works Bubeck et al. (2018b) and Bubeck et al. (2018a), henceforth referred to as BPR and BLPR respectively.[1] This is the starting point of our work.

**World** 1**.** No robust classifiers exist, regardless of computational or sample-efficiency considerations. Fawzi et al. (2018) show a learning task in this world, namely one where computationally efficient non-robust classification is possible, no robust classifiers exist. On the other hand, for natural learning tasks, humans seem to be robust classifiers that tolerate non-zero error rate $\epsilon$, indeed even efficient robust classifiers; see Bubeck et al. (2018b) for a more detailed discussion.

**World** 2**.** Robust classifiers exist, but they are computationally inefficient. We demonstrate learning tasks in this world.

**World** 3**.** Computationally efficient robust classifiers exist, but learning them incurs large sample complexity. Schmidt et al. (2018) show a learning task where a computationally efficient robust classifier exists, but learning it requires polynomially more samples than non-robust learning. On the other hand, Bubeck et al. (2018b) show that this gap *cannot be* more than linear in the dimension; see Bubeck et al. (2018b) for a more detailed discussion.

**World** 4**.** Computationally efficient robust classifiers exist, and can be learned sample-efficiently, but training is computationally inefficient. Bubeck et al. (2018a) show a learning task in this world. However, as we observe below, their *computationally efficient* robust classifier only recovers from a very small *number* (indeed, a constant number) of perturbations. Whether there exists an efficient robust classifier for their task that recovers from large perturbations seems related to long-standing open questions in computational number theory Heninger (2019); Green (2013). As our second result, we show two examples of learning tasks that live in this world; more details in Section 2.

1. To be precise, Bubeck et al. (2018b) postulated four worlds, namely worlds 1 and 3–5. Subsequent work of Bubeck et al. (2018a) added the second world.

**World 5.** The best world of all, in which there exist efficient algorithms both for classification and training, and the sample complexity is small (but it could be that we haven't discovered the right algorithm just yet.)

We want to understand – *are we likely to find learning tasks such as the ones Bubeck et al. (2018a) and we demonstrate in the wild?* To that end, our third result is a *win-win* statement: namely, any such learning task gives rise to a cryptographic object– either a simple one like a one-way function or a complex one like public-key encryption.

We proceed to describe the three results in more detail.

But before we do so, a word of warning. We and Bubeck et al. (2018a) define these five worlds in a coarse way using polynomial-time as a proxy for computational efficiency, and a large constant accuracy as a proxy for successful classification. (We should also mention that Bubeck et al. (2018b) use SQ-learning as a different proxy for computationally efficient learning.) One could be more careful and speak of running-time/accuracy tradeoffs in the different worlds, but since our goal here is to show broad counterexamples, we do not attempt to do such a fine-grained distinction.

## 2. Our Results

We explore the relationship of computational constraints and efficient robust classification. The setting we consider consists of two distributions $D_0, D_1$ and the classifier has to correctly classify inputs from both. We consider the two facets to efficient robust classification: (1) *existence:* do efficient robust classifiers exist? (corresponds to World 2) and (2) *learnbility:* can we learn robust classifiers efficiently? We show three sets results on which we elaborate below.

### 2.1. Existence (World 2)

In terms of feasibility, we show that there are learning tasks where while inefficient robust classification is possible, *no efficient robust classifiers exist*. That is, we demonstrate learning tasks in World 2. We can show the following:

**Theorem 1 (Informal)** *There exist classification tasks over where (1) efficient non-robust classifiers exist, (2) no efficient robust classifier exists, but (3) inefficient robust classifiers exist.*

This result does not require cryptographic assumptions, and relies only on the existence of *average-case hard* functions and good error-correcting codes. In fact, this result scales down to more fine-grained notions of efficiency than polynomial-time. All that is required is a function that is average-case hard for the "efficient" class, but computable by the "inefficient" class.

We give several examples of such learning tasks, including some examples that require cryptographic assumptions but obtain other desirable properties (such as obtaining tasks with efficiently-samplable distributions). More details are given in Sections 3.4 and 3.5 and Appendices B, C and E.

### 2.2. Learnability (World 4)

We want to understand the hardness of learning an efficient robust classifier when it exists. The starting point of this work was the BLPR work (Bubeck et al., 2018a). They showed that under cryptographic assumptions, there exists a learning task which admits efficient robust classifiers,

but it is computationally infeasible to train such a classifier. More precisely, they showed that there exists a classification task (over $\{0,1\}^n$) where (a) learning *any non-trivial* robust classifier is computationally infeasible while (b) an efficient robust classifier exists.

Unfortunately, we observe that their robust classifier is efficient only when correcting a constant number of errors. Indeed, as we explain in Section 3.1, the question of whether there exists a computationally efficient robust classifier for their task correcting even $\omega(1)$ bits of error is an important open question in computational number theory that has received some attention in the cryptanalysis community (Green, 2013; Heninger, 2019).

The BPLR construction can be rescued using error correcting codes to enable efficient robust classifiers robust to large (constant fraction) perturbations. Our results strengthen theirs in two ways: we can weaken the required cryptographic assumption to that one-way functions exist and demonstrate tasks where the gap between learning and robust classification is more: in that efficient learning algorithms can learn to not only classify, but also to generate fresh samples from the distributions.

**Theorem 2 (Informal)** *Under the minimal cryptographic assumption that one-way functions, there exist classification tasks over $\{0,1\}^m$ where (1) it is easy to learn a non-robust classifier (2) an efficient robust classifier that tolerates $m/8$-sized perturbations exists, and (3) it is computationally hard to learn any non-trivial robust classifier.*

**Theorem 3 (Informal)** *Assuming Learning Parity with Noise (or Learning with Errors) in the "public-key" regime of parameters, there exist classification tasks on $\{0,1\}^m$ where (1) it is easy to learn a non-robust classifier. (2) an efficient robust classifier tolerating $O(\sqrt{m})$-errors exists, and (3) it is computationally hard to learn any non-trivial robust classifier.*

*Furthermore, it is easy to learn generators/evaluators for the non-robust distributions.*[2]

We elaborate on the differences between the two theorems in the techniques section. Briefly, there are three key differences: Theorem 3 requires a stronger assumption, but gives a more "natural" example where the resulting distributions are "more easier" to learn non-robustly. In particular, it is easy to learn how to generate fresh samples from the two distributions, something that the one-way function based example cannot support. This is important because we want to separate the complexity of learning the distribution from that of robust classification. And here, these distributions can be learned in a stronger sense while still being hard to classify under adversarial perturbations.

### 2.3. A Win-Win Result

Finally, we want to understand – *Are we likely to find such learning tasks in the wild?* To that end, we show a converse to our results. Namely,

**Theorem 4 (Informal)** *Any computational task where an efficient robust classifier exists, but is hard to learn one in polynomial time implies one-way functions, and hence symmetric key cryptography.*

2. Generators and Evaluators (Kearns et al., 1994), are algorithms that can sample from the distribution and output the pdf of the distribution respectively.

Furthermore, if the learning task satisfies certain natural properties, it gives us (a certain weaker form of) public-key cryptography as well!

It would be very surprising to us if public-key cryptography (and even one-way functions) arise out of natural classification tasks on, say, images. Thus, perhaps uncharacteristically for cryptographers, we offer a possible (optimistic) interpretation of this state of affairs: namely, that for *natural* learning tasks where there exists a robust classifer, it can also be efficiently found, we just haven't figured out the right algorithm yet.

An important caveat is due here: our definition of hardness of learning a robust classifier is a strong one: it requires that the perturbing adversary be constructive and universal. Our classification tasks do satisfy this definition, and that only makes them stronger. On the other hand, it does make our converse weaker. More details are given in Section 3.

### 2.4. Related Work.

The works closest to ours are Bubeck et al. (2018b,a). We discuss them last.

***Adversarial Examples.*** The problem of adversarial classification ws first considered by Dalvi et al. (2004). Starting with Szegedy et al. (2013), there is a large body of work demonstrating the existence of small adversarial perturbations in neural networks that cause them to misclassify examples with high confidence. There have been various approaches proposed against such perturbations and many of them have been broken (see Carlini and Wagner (2017); Athalye et al. (2018) and references therein).

A line of work which (Gilmer et al., 2018; Fawzi et al., 2018; Mahloujifar and Mahmoody, 2018) shows that for certain learning tasks and distributions (eg spheres in $\mathbb{R}^n$ or product distributions), due to concentration of measure adversarial examples exist close to points in the distribution and can at times be found efficiently for classifiers that are not perfectly correct, pointing to the challenges of robust classification in this setting. These works show evidence for World 1: that for certain specific models and training algorithms, robust classifiers don't exist. In our learning tasks, robust classification is possible, albeit computationally inefficient.

Schmidt et al. (2018) demonstrate simple classification tasks (distinguishing between high dimensional gaussians) where the sample complexity of robust learning is higher than that of classical learning by a polynomial factor. Hence they show evidence for world 3. Bubeck et al. (2018b) show that this gap is essentially tight. This work is similar in spirit to ours, with the resource being sample complexity instead of computational complexity in our case. In the case of computational complexity, we can essentially show exponential gap between the running time required for learning non-robustly vs learning robust classifiers.

***BPR/BLPR (Bubeck et al., 2018b,a).*** In BPR, they showed two results. First, that in the world of polynomial sample complexity with no bounds on running time, learning a non-robust classifier and learning a robust classifer have the comparable sample complexity, if such a robust classifier exists. Second, they exhibit a learning task where while learning a robust classifier was information-theoretically easy with polynomial sample complexity, but doing so was difficult in the SQ model and it required exponentially many queries. This gives rise to a task where learning a robust classifier in a computationally efficient manner (in the SQ model) was a lot harder than doing so inefficiently.

In a followup work, BLPR they considered strengthening the second BPR result to show that under cryptographic assumptions, there exists a learning task which admitted efficient robust classifiers, but it was computationally infeasible to do so. They showed that there exists a classification

task (over $\{0,1\}^n$) where learning any non-trivial robust classifier is computationally infeasible while an efficient robust classifier exists that can correct $O(1)$-bit error. A description of their construction is given in Section 3.1 and Appendix G.

## 3. Our Techniques

In this section, we give a high level description of our techniques. We begin by describing the BLPR classification task and its limitations. Then we describe the definition of robust classification and non-existence/unlearnability of such classifiers. We then describe several recipes for constructing tasks where robust classification is computationally intractable. In the first recipe, based on one-way functions, we show tasks where while efficient robust classifiers exist, but are hard to learn, thus proving Theorem 2. The second recipe assuming average-case hard functions proves Theorem 1, where no efficient robust classifiers exist. The final recipe is based on hardness assumptions on decoding noisy codewords / lattices, namely Learning Parity with Noise (LPN) and Learning with Errors (LWE) and proves Theorems 1 and 3 in different parameter regimes.

### 3.1. The BLPR Classification Task

We sketch the Bubeck et al. (2018a) classification task where it is difficult to learn a robust classifier. A more detailed description of their construction is given in Appendix G.

The key object in their construction is a "trapdoor pseudorandom generator". A pseudorandom generator $\mathsf{PRG} : \{0,1\}^n \to \{0,1\}^{2n}$ is an expanding function whose outputs are indistinguishable from truly random strings. That is, $\{\mathsf{PRG}(x) : x \leftarrow \{0,1\}^n\} \approx_c \{y : y \leftarrow \{0,1\}^{2n}\}$.[3] A trapdoor pseudorandom generator has a hard-to-find trapdoor $\mathsf{trap}$ that allows distinguishing the output of the PRG from random outputs. That is, there exists a distinguisher $\mathsf{D}$ such that (say),

$$\mathop{\mathbb{P}}_{x\leftarrow\{0,1\}^n}[\mathsf{D}(\mathsf{trap},\mathsf{TrapPRG}(x))=1] - \mathop{\mathbb{P}}_{y\leftarrow\{0,1\}^{2n}}[\mathsf{D}(\mathsf{trap},y)=1] > 0.99$$

They show that the Blum-Blum-Shub Pseudorandom generator (Blum et al., 1986) has such a trapdoor. Given a trapdoor PRG, their learning task $D_0, D_1$ is the following:

$$D_0 = \{(0,\mathsf{TrapPRG}(x)) : x \leftarrow \{0,1\}^n\} \text{ and, } D_1 = \left\{(1,y) : y \leftarrow \{0,1\}^{2n}\right\}.$$

The first bit enables easy non-robust classification. The fact that there exists an inefficient robust classifier follows from a volume argument – that the there are a few PRG outputs in a large domain. This implies that there is an inefficient robust classifier that tolerates $O(\sqrt{n})$-sized perturbations. That a robust classifier is hard to learn follows from the perturbing adversary that sets the first bit to 0. A robust classifier has to distinguish between outputs of the PRG from random strings, without the trapdoor. This is infeasible by the security guarantee of the PRG.

Finally, what needs to be proved is that the trapdoor enables *robust classification*. The trapdoor indeed does enable a robust classifier that tolerates constant-sized pertubations (i.e., if any constant number of bits are altered) simply by exhaustive search among the polynomially many possible sets of perturbed bits. For a constant $c$, the robust classifier given input $y$ goes over all $n^c$ words in the

3. We say that two families of distributions $\{X_n\}_{n\in\mathbb{N}}$ an $\{Y_n\}_{n\in\mathbb{N}}$ are computationally indistinguishable (denoted by $\{X_n\} \approx_c \{Y_n\}$ or $X \approx_c Y$ for brevity) if for every polynomial time distinguisher $\mathsf{D}$, $|\mathbb{P}_{x\leftarrow X_n}[\mathsf{D}(x)] - \mathbb{P}_{y\leftarrow Y_n}[\mathsf{D}(y)]| \le \mathsf{negl}(n)$.

Hamming ball $y' \in B(y, c)$ and checks if the distinguisher $\mathsf{D}(\mathsf{trap}, y') = 1$. If yes, output $D_0$ else output $D_1$. But this approach does not give a classifier beyond constant-sized errors because the running time is exponential in the number of errors corrected.

The primary limitation of trapdoor PRGs is that the trapdoor does not enable decoding the PRG output from the perturbed samples, only distinguishes PRG outputs from random strings. Indeed, for the Blum-Blum-Shub trapdoor PRG (and related constructions such as the one of Micali and Schnorr (Micali and Schnorr, 1991)) considered in BLPR, the question of whether there is any trapdoor that permits robust inversion is an open question in computational number theory. We refer the reader to Appendix G for discussions regarding related questions. To enable efficient decoding, their construction can be modified by using an error correcting code to make it robust to larger pertubations.

### 3.2. Definitions: Robust Classification

We start by describing the notion of robust classification and hardness of robust classification used.

**Robust Classifier.** When we state that *a robust classifier* exists (for given $\varepsilon$), we show the strongest notion: that there exists a classifier $\mathsf{R}$ (efficient or inefficient, as specified) that classifies all input close to a random sample correctly:

$$\text{For } b \in \{0,1\}, \underset{x \leftarrow D_b}{\mathbb{P}}\left[\mathsf{R}(x') = b \text{ for all } x' \in B(x, \varepsilon)\right] > 0.99 \,.$$

**Non-Existence/Unlearnability of Robust Classifiers.** When we describe *the non-existence (or unlearnability) of robust classification*, we satisfy the strongest notion: that there exists a poly-time perturbation adversary $\mathsf{P}$ whose perturbed examples cannot be classified better than chance by any efficient (or efficiently learned) classifier. That is, for any efficient $\mathsf{R}$ (or $\mathsf{R} \leftarrow \mathsf{learn}^{D_0, D_1}(1^n)$),

$$\underset{x \leftarrow D_b}{\mathbb{P}}\left[\mathsf{R}(\mathsf{P}^{D_0, D_1}(x)) = b\right] < 0.5 + \mathsf{negl}(n) \,,$$

where a $\mathsf{negl} : \mathbb{N} \to \mathbb{R}$ is a function such that $\mathsf{negl}(n) < n^{-c}$ for all $c \in \mathbb{N}$ for large enough $n$. See Definition 10 for a formal definition of hard to learn robustly. This definition has two key properties: it is *constructive* (adversarial perturbations are found) and *universal* (the same adversary works for all algorithms).

The negation of the robust classification definition suggests the following definition: for efficiently learned classifiers $\mathsf{R}$, $\mathbb{P}_{x \leftarrow \mathcal{D}_b}\left[B(x, \varepsilon) \not\subseteq \mathsf{R}^{-1}(b)\right] < 0.99$. This definition is unsatisfying because it says nothing about the hardness of finding such misclassified examples. In particular, if such adversarial perturbations existed but were computationally hard to find, then the existence of adversarial examples is not an issue. Hence, we choose a constructive definition that requires such examples to be efficiently found. The fact that the adversary is universal only makes the counter examples stronger.

### 3.3. Unlearnability From Pseudorandom Functions and Error Correcting Codes

In this section, we construct a learning task where classification is easy, robust classifier exits, but is hard to learn. The primary ingredients of this construction are pseudorandom functions and error correcting codes. We introduce both the primitives and build the construction in stages. A pseudorandom function family (PRF) (Goldreich et al., 1986) is a family of keyed functions $F_k$ :

$\{0,1\}^n \to \{0,1\}$ where the key $k \leftarrow \{0,1\}^n$, that are indistinguishable from uniformly random functions to any polynomial time algorithm. That is, for every poly time algorithm $\mathsf{A}$,

$$\mathop{\mathbb{P}}_{k\leftarrow\{0,1\}^n}\left[\mathsf{A}^{F_k}(1^n)\right] \approx \mathop{\mathbb{P}}_{U_n}\left[A^{U_n}(1^n)\right]$$

where $U_n : \{0,1\}^n \to \{0,1\}$ is a uniformly random function. PRFs can be constructed from one-way fucntions. Kearns and Valiant (Kearns and Valiant, 1994) constructed a hard to learn classification task using pseudorandom functions as follows:

$$D_0 = (x, F_k(x)) \text{ and, } D_1 = (x, 1 - F_k(x))$$

The task essentially asks to efficiently learn a predictor for the pseudorandom function which is difficult. To transform this task to one that is hard to learn robustly, while an efficient robust classifier exists, we use error correcting codes. Recall that an error correcting code has two algorithms $(\mathsf{Encode}, \mathsf{Decode})$ where $\mathsf{Encode}$ returns a redundant encoding of the message that the $\mathsf{Decode}$ algorithm can efficiently recovers the encoded message even when the encoded codeword is tampered adversarially to some degree. So, consider the following classification task: distinguish between error-corrected versions of the PRF:

$$D_0 = \mathsf{Encode}(x, F_k(x)) \text{ and, } D_1 = \mathsf{Encode}(x, 1 - F_k(x)) \ .$$

Note that this task has the following properties: (1) A robust classifier exits and, (2) a robust classifier is hard to learn. For the first property, consider the following robust classifier: the classifier given the secret key, first decodes the perturbed sample using the $\mathsf{Decode}$ algorithm and then checks if is of the form $(x, F_k(x))$ or $(x, 1 - F_k(x))$ and outputs which case it is. The robustness follows from the error correcting code. The fact that no classifier is learnable follows from the fact that the PRF is hard to predict, and thats exactly what the classifier has to do. Finally, we want the task to be easy to classify non-robustly. Here we use the "BPR trick" (Bubeck et al. (2018b)). That is, we additionally append to each sample a bit indicating which distribution it was sampled from. That is,

$$D_0 = (0, \mathsf{Encode}(x, F_k(x))) \text{ and, } D_1 = (1, \mathsf{Encode}(x, 1 - F_k(x))) \ .$$

Now the samples are easy to classify non-robustly, simply output the first bit. Learning a robust classifier is hard, for that, consider the perturbing adversary that erases the first bit. For these samples, robust classification is identical to predicting the output of the PRF. This is difficult for any efficiently learned classifier. Hence, this gives us a task that that is easy to classify, has an efficient robust classifer and yet, any non-trivial robust classifier is hard to learn.

Note that because we have excellent error correcting codes, this recipe is maximally robust. We can pick a code that tolerates a constant fraction $(\frac{1}{4} - \varepsilon)$ errors and still enable correct decryption (Guruswami and Indyk, 2001). This can be further boosted to $(\frac{1}{2} - \varepsilon)$ by using list decoding instead of unique decoding and increasing the output size of the PRF to $n$-bits. We do not formally write this construction.

### 3.4. Non-Existence of Robust Classifiers from Average-Case Hardness

This section describes a learning task for which no computationally efficient robust classifier exists, even though inefficient ones do, based on average-case hard functions, thus proving Theorem 1.

Let $g : \{0,1\}^n \to \{0,1\}$ be a function that is *average-case hard*, such that no polynomial-time nonuniform algorithm can compute $z \mapsto g(z)$ noticeably better than random guessing. For example, taking $g$ to be a random function $\{0,1\}^n \to \{0,1\}$ suffices. Let $(\mathsf{Encode}, \mathsf{Decode})$ be a good error correcting code, capable of decoding from a constant fraction of errors. Now, construct distributions $D_0, D_1$ as follows:

$$D_0 = (0, \mathsf{Encode}(x, g(x))) \text{ and } D_1 = (1, \mathsf{Encode}(x, 1 - g(x)))$$

for $x \leftarrow \{0,1\}^n$ uniformly. Note that these distributions are trivially distinguishable non-robustly. However, with a perturbation adversary that destroys the first coordinate, distinguishing $D_0$ from $D_1$ essentially requires computing the function $g(x)$, which cannot be done efficiently. Thus, there is no efficient robust classifier. Moreover, an inefficient robust classifier exists, since one can decode the error correcting code (correcting any adversarial errors) and compute $g(x)$.

When using an average-case hard function, one limitation here is that the algorithm generating the samples from distributions $D_0, D_1$ is inefficient. This can be remedied by using one-way functions, because generating $D_0, D_1$ requires the algorithm to perform the simpler task of sampling $(z, g(z))$ for random $z$'s, and not computing $g(z)$ given $z$, that the classifier has to do. In fact, this is precisely the difference between average-case $\mathsf{NP}$ hardness, which requires us to generate hard instances, and one-way functions, which require generating hard instances along with their solutions. See Section 3.4 for more details.

### 3.5. From Hardness of Decoding under Noise.

This section describes a proof sketch for Theorems 1 and 3. The problems Learning Parity with Noise (LPN) and Learning with Errors (LWE) have the following flavor: In both the problems a random code $C$ (over $\mathbb{Z}_2$ in LPN, $\mathbb{Z}_q$ for a large prime in LWE) is specified by a matrix $\mathbf{A}$:

$$C = \{\boldsymbol{s}^T \mathbf{A}\} \text{ or, the dual form } C = \{\boldsymbol{y} : \mathbf{A}\boldsymbol{y} = \mathbf{0}\} .$$

Then the computational task is to distinguish a point close to the code from a uniformly random point in the space. The conjectured hardness of these problems can be used to construct a variety of cryptographic primitives. In the overview, we will describe the construction with the LPN assumption. The LWE construction is conceptually identical.

**The Classification Task.** We begin by describing the classification task and then the rationale. The task consists of two distributions on samples $D_0, D_1$ picked as follows: Pick a random linear code over $\mathbb{Z}_2$, $C : \{0,1\}^n \to \{0,1\}^{8n}$, (described by the generator matrix $\mathbf{A}$ or the parity check matrix $\mathbf{H}$). Then,

$$D_0 = \{\boldsymbol{y} : \boldsymbol{y} \leftarrow C\} \text{ and, } D_1 = \{\boldsymbol{y} + \mathbf{1} : \boldsymbol{y} \leftarrow C\} .$$

So, the task is to distinguish codewords of $C$ from their affine shift ($\mathbf{1}$ represents the all-ones vector). The distributions are easy to classify non-robustly. There exists an inefficient robust classifier because the distance between the two codes $C$ and $C + \mathbf{1}$ is large.

To show that a robust classifier is hard to learn, consider the perturbation adversary that adds random noise of varying size to the two distributions. Learning a robust classifier for this adversary is equivalent to distinguishing LPN samples from random. Hence any computationally efficient adversary cannot classify these examples better than chance.

Finally, we need to show that for a certain perturbation regime, no efficient robust classifier exists while for a different perturbation regime, an efficient robust classifier does exist. The latter is accomplished by the notion of "trapdoor sampling" where the code is sampled with a trapdoor that enables decoding noisy codewords (and hence robust classification too).

Below we describe the example in more detail and give a sketch of the arguments needed. Formal proofs are given in Appendices B and C.

**LPN Assumption.** The LPN hardness assumption states that: for $m = \mathsf{poly}(n)$,

$$(\mathbf{A}, \boldsymbol{s}^T\mathbf{A} + \boldsymbol{e}^T) \bmod 2 \approx_c (\mathbf{A}, \boldsymbol{r}) \bmod 2$$

where $\mathbf{A} \leftarrow \mathbb{Z}_2^{n\times m}$ describes a random code, the secret $\boldsymbol{s} \leftarrow \mathbb{Z}_2^n$ is drawn unifomly at random and each coordinate of error $\boldsymbol{e} \leftarrow \mathrm{Ber}(r)^m$ drawn from a Bernoulli distribution with error rate $r$, i.e., probability of drawing 1 is $r$.

**Hardness Regimes and Trapdoors.** The key parameter which controls the hardness of LPN is the distance of the close point from the code in the appropriate norm.[4] As the distance increases, the problem becomes harder. In the case of LPN, this distance is approximately $m \cdot r$ where $r$ is the error rate. For most non-trivial parameter settings of the distance parameter, these two problems are *believed to be computationally intractable*. That is, an efficient algorithm given a description of the random code cannot distinguish random points close to the code from random points in the space.

Along with their conjectured computational hardness, we are interested in another property of these problems, *the existence of a trapdoor*: that is, can we sample the code along with some polynomial-size side information that lets us distinguish efficiently random points from points close to the code. This information usually is a "short basis" for the dual code. The trapdoor property has two important regimes: the "public-key" regime and the "private-key" regime. In the case of LPN, the public-key regime corresponds to error rate $r = O(1/\sqrt{n})$ while the private-key regime translates to constant error rates, e.g., $r = 0.1$. The public key regime of parameters enables construction of advanced cryptographic primitives, including public key encryption. On the other hand, in the private-key regime, we know constructions of one-way functions and symmetric key cryptography, but not much more.

Importantly for us, in "public-key" parameter regime, such a trapdoors exists and can be sampled efficiently. On the other hand, in the private-key regime, it is conjectured that no such trapdoor exists. Traditionally this problem is studied as the problem of *decoding linear codes with preprocessing* (for LPN) and *closest vector problem with preprocessing* (for LWE). In the problem of decoding linear codes with preprocessing, an inefficient algorithm $\mathsf{Preprocess}$ performs arbitrary preprocessing on the given linear code (described by the matrix $\mathbf{A}$) and has to come up with a short polynomial-sized trapdoor for the code. Later the $\mathsf{Decode}$ algorithm has to use this trapdoor to efficiently find the codeword close to a given input. This problem and the closest vector problem (is the same problem, on lattices instead of codes) are $\mathsf{NP}$-hard to approximate in the worst-case (Bruck and Naor, 1990; Micciancio, 2001; Regev, 2004).

We require an average-case variant of the problem termed as the hardness of *LPN with Preprocessing*. The assumption is stated more formally in Assumption 14. This assumption can be used to construct a task where no efficient robust classifier exists. The task is very similar to the one below

4. The specific norm is not crucial for the discussion below. Hamming is used for LPN while for LWE, the norm is obtained by embedding $\mathbb{Z}_q$ in $\mathbb{Z}$ as $\{-\lfloor q/2\rfloor, \ldots 0, 1, \ldots \lfloor q/2\rfloor\}$ and take the $\ell_\infty$ norm on $\mathbb{Z}$.

where a efficient robust classifier exists but is hard to find, except with higher levels of noise. More details in Appendix B.

**Task with an Hard-to-Learn Efficient Robust Classifier.** We now turn to the problem of constructing learning tasks where an efficient robust classifier exists, but is hard to learn. It consists of distinguishing between codewords ($D_0 = \{\boldsymbol{s}\mathbf{A}\}$) from an affine shift of the codewords ($D_1 = \{\boldsymbol{s}\mathbf{A} + \mathbf{1}\}$ where $\mathbf{1}$ is the all-ones vector). That is, for a random matrix $\mathbf{A} \in \mathbb{Z}_2^{n\times m}$ where $m = 8n$,

$$D_0 = \{\boldsymbol{s}^T\mathbf{A} : \boldsymbol{s} \in \{0,1\}^n\} \text{ and, } D_1 = \{\boldsymbol{s}^T\mathbf{A} + \mathbf{1} : \boldsymbol{s} \in \{0,1\}^n\}$$

We want to show that this task exhibits an efficient robust classifier. For that, we need access to a trapdoor. In the case of LWE, such algorithms are known (Gentry et al., 2008) and proven to be exremely fruitful (see Peikert (2016) and references therein). This LWE trapdoor is a zero-one matrix $\mathbf{T} \subseteq \{0,1\}^{m\times m-n}$ over $\mathbb{Z}_q$ such that $\mathbf{AT} = \mathbf{0} \pmod q$. Because $\mathbf{T}$ is a zero-one matrix, given any adversarially perturbed sample $\tilde{\boldsymbol{y}} = \boldsymbol{s}^T\mathbf{A} + \boldsymbol{e}^T$, multiplying by $\mathbf{T}$ results in $\tilde{\boldsymbol{y}}\mathbf{T} = \boldsymbol{e}^T\mathbf{T}$ which is a vector with small entries in each coordinate. And this can be checked to distinguish LWE samples from random.

In the case of LPN, we don't know how to perform such trapdoor sampling: where a uniformly random matrix $\mathbf{A}$ is sampled along with such a trapdoor. Instead we rely on a computational variant of this. We can sample a matrix $\mathbf{H} \in \mathbb{Z}_2^{n\times 8n}$ that is indistinguishable from a random matrix along with such a "short" trapdoor: a matrix $\mathbf{E}$ where each row and colum of $\mathbf{E}$ has hamming weight $O(\sqrt{m})$. See Theorem 15 for more details. This then allows for a similar construction. The perturbing adversary $\mathsf{P}$ again adds random noise, this time of a lower magnitude though.

$$\mathsf{P}(\boldsymbol{y}) : \text{Output } \tilde{\boldsymbol{y}} = \boldsymbol{y} + \boldsymbol{e} \text{ where } \boldsymbol{e} \leftarrow \mathbb{Z}^m_{2,\mathsf{Ham}=0.1\sqrt{m}} \,.$$

Note that earlier, we added $0.1m$ bits of noise, instead here we are adding $0.1\sqrt{m}$ bits. This level of noise places the problem in the "public-key" regime of parameters. Furthermore, given the trapdoor, in this case, we can recover which distribution the unperturbed sample was sampled from, giving us the required robust classifier. See Appendix B for more details.

Again, it is clear that learning a non-robust classifier is easy. The hardness of LPN assumption implies that it is hard to learn a robust classifier. This is in contrast to the previous construction where no efficient robust existed. Here, the trapdoor gives us an efficient robust classifier, but the hardness of LPN implies that such a classifier is hard to learn. In fact, any efficiently learned classifier cannot do better than chance.

One thing to note is that this efficient robust classifier is not "maximally robust", meaning that while an inefficient robust classifier can tolerate $0.1m$ bits of noise and still classify correctly, the efficient classifier can tolerate $0.1\sqrt{m}$ bits of noise. This is similar in the case of LWE as well, where there is a gap between noise the trapdoor can support (about $q/m$ in the $\ell_\infty$ norm) against the maximally robust limit ($\Omega(q)$). This is not surprising because decoding random linear codes is harder than decoding specifically designed codes and hence the trapdoors do not achieve optimal decoding.

A feature of this construction is that an efficient algorithm can learn to not only distinguish the samples from distributions $D_0$ and $D_1$, it can easily learn to generate samples from the two distributions as well.

**Comparing Recipes.** There are three key differences between the recipes. The first difference is in the underlying hardness assumption. The first two constructions are based on weaker assumptions: namely general assumptions that one-way functions exist (or average-case hard function respectively) rather than the specific assumptions of LWE and LPN.

The second difference is that the distributions based on LWE/LPN facilitate learning in a stronger sense, that it is possible to sample from the non-robust distributions after seeing polynomially many samples. In construction I based on one-way functions, we do not learn either $D_0$ or $D_1$ in that strong sense. In fact, after seeing polynomially many samples, efficient sampling algorithms have no non-trivial advantage with the other recipes. As pointed out in in construction II, it is possible to support generation, albeit using a slightly stronger assumption that one-way functions exist.

The third difference is that of naturalness: we feel that the LWE/LPN recipe gives a more natural learning task. This is obviously a subjective notion. This learning task of distinguishing noisy codewords from random has existed independent of the notion of robust classification and arises naturally in other contexts.

### 3.6. Converse: Cryptography from Hardness of Robust Classification.

In this section, we describe how Theorem 4 is proved. The key result we rely on here is that we can construct one-way functions from any pair of samplable distributions that are statistically far and computationally indistinguishable.

**Theorem 5** *Given a pair of distributions $(X_0, X_1) \leftarrow \mathcal{F}$ over $\mathcal{X}$ that are statistically far, i.e., $d_{TV}(X_0, X_1) > 0.9$ and computationally indistinguishable. That is for every polynomial time adversary $\mathsf{A}$ that gets sample access to the distributions,*

$$\mathop{\mathbb{E}}_{x \leftarrow X_0} \mathsf{A}^{(X_0,X_1)}(x) - \mathop{\mathbb{E}}_{x \leftarrow X_1} \mathsf{A}^{(X_0,X_1)}(x) < 0.1$$

*Then one-way functions exist.*[5]

In order to construct such distributions, we rely on the learning task (given by $(D_0, D_1)$) and the perturbation adversary $\mathsf{P}$. The distributions we consider are

$$X_0 = \{\mathsf{P}(x) : x \leftarrow D_0\} \text{ and, } X_1 = \{\mathsf{P}(x) : x \leftarrow D_1\} .$$

Note that because efficient robust classifiers are hard to learn, no efficient algorithm $\mathsf{A}$ (that knows $\mathsf{P}$ and gets access to the distributions $D_0, D_1$) can distinguish between the two distributions . On the other hand, because a robust classifier exists, these two distributions are statistically far from each other. This implies that one-way functions exist.

**Acknowledgments.** We would like to thank Shafi Goldwasser and Nadia Heninger for discussions regarding inversion of the (noisy) BBS PRG.

5. The constants in the equations are fairly arbitrary. We can replace them by any constants $\alpha, \beta$ where $\alpha > \beta$ and the result holds (see Naor and Rothblum (2006); Berman et al. (2019)).

## References

Michael Alekhnovich. More on Average Case vs Approximation Complexity. In *Foundations of Computer Science*, pages 298–307. IEEE, 2003.

Sanjeev Arora and Rong Ge. New algorithms for learning in presence of errors. In *International Colloquium on Automata, Languages, and Programming*, pages 403–415. Springer, 2011.

Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. *arXiv preprint arXiv:1802.00420*, 2018.

Itay Berman, Akshay Degwekar, Ron D. Rothblum, and Prashant Nalini Vasudevan. Statistical Difference Beyond the Polarizing Regime. *Electronic Colloquium on Computational Complexity (ECCC)*, 26:38, 2019. URL https://eccc.weizmann.ac.il/report/2019/038.

Battista Biggio and Fabio Roli. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition*, 84:317–331, 2018.

Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant Learning, the Parity Problem, and the Statistical Query Model. *J. ACM*, 2003.

Lenore Blum, Manuel Blum, and Mike Shub. A Simple Unpredictable Pseudo-Random Number Generator. *SIAM Journal on computing*, 1986.

Jehoshua Bruck and Moni Naor. The Hardness of Decoding Linear Codes with Preprocessing. *IEEE Trans. Information Theory*, 36(2):381–385, 1990. doi: 10.1109/18.52484. URL https://doi.org/10.1109/18.52484.

Sébastien Bubeck, Yin Tat Lee, Eric Price, and Ilya P. Razenshteyn. Adversarial examples from cryptographic pseudo-random generators. *CoRR*, abs/1811.06418, 2018a. URL http://arxiv.org/abs/1811.06418.

Sébastien Bubeck, Eric Price, and Ilya Razenshteyn. Adversarial examples from computational constraints. *CoRR*, abs/1805.10204, 2018b. URL https://arxiv.org/abs/1805.10204.

Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 39–57. IEEE, 2017.

Nilesh Dalvi, Pedro Domingos, Sumit Sanghai, Deepak Verma, et al. Adversarial classification. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 99–108. ACM, 2004.

Alhussein Fawzi, Hamza Fawzi, and Omar Fawzi. Adversarial vulnerability for any classifier. *CoRR*, abs/1802.08686, 2018. URL http://arxiv.org/abs/1802.08686.

Yoav Freund, Robert E Schapire, et al. Adaptive game playing using multiplicative weights. *Games and Economic Behavior*, 29(1-2):79–103, 1999.

Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for Hard Lattices and New Cryptographic Constructions. In *Symposium on Theory of computing*, pages 197–206. ACM, 2008.

Justin Gilmer, Luke Metz, Fartash Faghri, Samuel S. Schoenholz, Maithra Raghu, Martin Wattenberg, and Ian Goodfellow. Adversarial spheres, 2018.

Oded Goldreich. Foundations of Cryptography: Basic Tools, 2001.

Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to Construct Random Functions. *J. ACM*, 1986. doi: 10.1145/6490.6503. URL http://doi.acm.org/10.1145/6490.6503.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. corr (2015).

Matthew Green. A few more notes on NSA random number generators, 2013. URL https://blog.cryptographyengineering.com/2013/12/28/a-few-more-notes-on-nsa-random-number/.

Venkatesan Guruswami and Piotr Indyk. Expander-based Constructions of Efficiently Decodable Codes. In *IEEE Foundations of Computer Science*. IEEE, 2001.

Nadia Heninger. Personal communication, 2019.

Russell Impagliazzo. Hard-core distributions for somewhat hard problems. In *Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on*, pages 538–545. IEEE, 1995.

Abhishek Jain, Stephan Krenn, Krzysztof Pietrzak, and Aris Tentes. Commitments and efficient zero-knowledge proofs from learning parity with noise. In *ASIACRYPT*, 2012.

Michael Kearns and Leslie Valiant. Cryptographic Limitations on Learning Boolean Formulae and Finite Automata. *J. ACM*, 1994.

Michael Kearns, Yishay Mansour, Dana Ron, Ronitt Rubinfeld, Robert E Schapire, and Linda Sellie. On the Learnability of Discrete Distributions. In *ACM symposium on Theory of computing*, pages 273–282, 1994.

Daniel Lowd and Christopher Meek. Adversarial learning. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 641–647. ACM, 2005.

Vadim Lyubashevsky. The parity problem in the presence of noise, decoding random linear codes, and the subset sum problem. In *APPROX'05/RANDOM'05*, 2005.

Saeed Mahloujifar and Mohammad Mahmoody. Can adversarially robust learning leverage computational hardness? *CoRR*, abs/1810.01407, 2018. URL http://arxiv.org/abs/1810.01407.

Silvio Micali and Claus-Peter Schnorr. Efficient, Perfect Polynomial Random Number Generators. *Journal of Cryptology*, 3(3):157–172, 1991.

Daniele Micciancio. The hardness of the closest vector problem with preprocessing. *IEEE Transactions on Information Theory*, 47(3):1212–1215, 2001.

Preetum Nakkiran. Adversarial robustness may be at odds with simplicity. *arXiv preprint arXiv:1901.00532*, 2019.

Moni Naor and Guy N Rothblum. Learning to impersonate. In *Proceedings of the 23rd international conference on Machine learning*, pages 649–656. ACM, 2006.

Chris Peikert. A Decade of Lattice Cryptography. *Foundations and Trends® in Theoretical Computer Science*, 10(4):283–424, 2016.

Oded Regev. Improved Inapproximability of Lattice and Coding Problems With Preprocessing. *IEEE Trans. Information Theory*, 50(9):2031–2037, 2004. doi: 10.1109/TIT.2004.833350. URL https://doi.org/10.1109/TIT.2004.833350.

Ludwig Schmidt, Shibani Santurkar, Dimitris Tsipras, Kunal Talwar, and Aleksander Madry. Adversarially robust generalization requires more data. *arXiv preprint arXiv:1804.11285*, 2018.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

Salil Vadhan and Colin Jia Zheng. A uniform min-max theorem with applications in cryptography. In *Advances in Cryptology–CRYPTO 2013*, pages 93–110. Springer, 2013.

## Appendix A. Definitions

We use lowercase letters for values, uppercase for random variables, uppercase calligraphic letters (e.g., $\mathcal{U}$) to denote sets, boldface for vectors (e.g., $\boldsymbol{x}$), and uppercase sans-serif (e.g., $\mathsf{A}$) for algorithms (i.e., Turing Machines). We let $\mathsf{poly}$ denote the set all polynomials. A function $\nu\colon \mathbb{N} \to [0,1]$ is *negligible*, denoted $\nu(n) = \mathsf{negl}(n)$, if $\nu(n) < 1/p(n)$ for every $p \in \mathsf{poly}$ and large enough $n$. Given a random variable $X$, we write $x \leftarrow X$ to indicate that $x$ is selected according to $X$. Similarly, given a finite set $\mathcal{S}$, we let $s \leftarrow \mathcal{S}$ denote that $s$ is selected according to the uniform distribution on $\mathcal{S}$. For an algorithm $\mathsf{A}$, we denote by $x \leftarrow \mathsf{A}$ the experiment where $x$ is sampled by feeding a uniformly random input to $\mathsf{A}$ from its input domain. We say that two families of distributions $\{X_n\}_{n\in\mathbb{N}}$ an $\{Y_n\}_{n\in\mathbb{N}}$ are computationally indistinguishable (denoted by $\{X_n\} \approx_c \{Y_n\}$ or $X \approx_c Y$ for brevity) if for every polynomial time distinguisher $\mathsf{D}$,

$$|\mathop{\mathbb{P}}_{x\leftarrow X_n}[\mathsf{D}(x)] - \mathop{\mathbb{P}}_{y\leftarrow Y_n}[\mathsf{D}(y)]| \leq \mathsf{negl}(n)$$

### A.1. Learning & Classification

**Definition 6 (Classification)** *For a family of classification tasks $\mathcal{F}$ over $\mathcal{X}$ is easy to classify if there exists a learning algorithm that given $\mathsf{poly}(n)$ i.i.d. samples from a pair of distributions $(D_0, D_1) \in \mathcal{F}$ supported on $\mathcal{X}$, outputs an efficiently computable classifier $\mathsf{A} : \mathcal{X} \to \{0,1\}$ such that,*

$$\mathop{\mathbb{P}}_{X\leftarrow D_b}[\mathsf{A}(x) = b] \geq 0.99$$

We want to consider other notions of learning distributions as well, in order to make more refined distinctions between learning distributions. The following definition for learnability of discrete distributions is from Kearns et al. (1994).

**Definition 7** *For a distribution $D$ over a discrete domain $\mathcal{X}$,*

1. ***Generator.*** *A circuit $\mathsf{G} : \{0,1\}^m \to \mathcal{X}$ is an $\varepsilon$-*good generator *for $D$ if*

$$\mathrm{KL}(D \| \mathsf{G}(U)) \leq \varepsilon$$

   *where $\mathsf{G}(U)$ denotes the distribution obtained by evaluating $\mathsf{G}$ on a uniformly random input.*

2. ***Evaluator.*** *A circuit $\mathsf{E} : \mathcal{X} \to \mathbb{R}_{\geq 0}$ is an $\varepsilon$-*good evaluator *for $D$ if*

$$\mathrm{KL}(D \| \mathsf{E}) \leq \varepsilon$$

   *where $\mathsf{E}$ denotes the distribution obtained by sampling with probability density function $\mathsf{E}$.*

**Definition 8** *A class of distributions $\mathcal{F} = \{F_n\}$ over a discrete domain $\mathcal{X} = \{\mathcal{X}_n\}$ is $(\varepsilon, \delta)$-*efficiently learnable with a generator (or evaluator resp.) *if there exists a polynomial time algorithm* Gen *that given oracle access to any $D_n \in \mathcal{F}_n$ runs in time* $\mathsf{poly}(n, 1/\delta, 1/\varepsilon)$ *and outputs $\mathsf{G}$ (or $\mathsf{E}$ resp.) such that with probability $\geq 1 - \delta$ over the randomness of* Gen *and samples, $\mathsf{G}$ ($\mathsf{E}$ resp.) is an $\varepsilon$-good generator (evaluator resp.) of $D$.*

In our examples, we seek to find distributions where the gap between ease of learning the actual distributions and that of the adversarially perturbed distributions is maximized.

### A.2. Hardness of Efficient Robust Classification

We start by recalling the notion of robust classification. Then, we consider two ways of formalizing the difficulty of efficient robust classification: (1) no *efficiently computable* robust classifier exists, (2) an efficient robust classifer exists, but *it is hard to learn one efficiently*.

**Definition 9** *Consider a classification task given by two distributions $D_0, D_1$ over $\mathcal{X}^n$. Let $\|\cdot\|$ be a norm over the space $\mathcal{X}^n$ and $\varepsilon > 0$. Let $\mathsf{R} : \mathcal{X}^n \to \{0,1\}$ be a classifier. The classifier $\mathsf{R}$ is $\varepsilon$-*robust *if*

$$\mathbb{P}_{X \leftarrow D_b}[\mathsf{R}(\tilde{x}) = b \text{ for all } \tilde{x} \in B(x, \varepsilon)] \geq 0.99$$

In the definition above, $B(x, \varepsilon)$ is all points that are $\varepsilon$ distance from $x$ in the given norm. Here, we will generally be concerned with Hamming distance and the $\ell_1$ norm.

**Definition 10 (Hardness of Robust Classification)** *Consider a family of classification tasks, defined by two distributions $D_0$, $D_1$ over $\mathcal{X}^n$ sampled from a distribution over learning tasks* Samp. *Let $\|\cdot\|$ be a norm over the space $\mathcal{X}^n$ and $\varepsilon > 0$. We consider the following notions of difficulty of robust classification:*

1. ***No efficient $\varepsilon$-robust classifier exists.*** *There exists a polynomial-sized perturbation algorithm* $\mathsf{P}$*, such that for every polynomial sized classifier* $\mathsf{R}$*, the perturbed samples are hard to classify. That is,*

$$\mathbb{P}\left[\mathsf{R}^{D_0,D_1}(\tilde{x}) = b\right] \leq \frac{1}{2} + \mathsf{negl}(n)$$

*where the perturbed sample $\tilde{x}$ is generated by sampling $x \leftarrow D_b$ for a random $b \leftarrow \{0,1\}$ and is then perturbing $\tilde{x} \leftarrow \mathsf{P}^{D_0,D_1}(x)$.*

2. ***Efficient $\varepsilon$-robust classifier is hard to learn.*** *There exists a polynomial-sized perturbation algorithm* $\mathsf{P}$*, such that every polynomial-time learning algorithm* $\mathsf{learn}$ *that outputs a polynomial sized classifier* $\mathsf{R}$*, the perturbed samples are hard to classify for* $\mathsf{R}$*. That is, for a learning task $D_0, D_1$ sampled by* $\mathsf{Samp}$ *and robust classifer* $\mathsf{R} \leftarrow \mathsf{learn}^{D_0,D_1}(1^n)$ *output by* $\mathsf{learn}$*,*

$$\mathbb{P}[\mathsf{R}(\tilde{x}) = b] \leq \frac{1}{2} + \mathsf{negl}(n)$$

*where the perturbed sample $\tilde{x}$ is generated by sampling $x \leftarrow D_b$ for a random $b \leftarrow \{0,1\}$ and is then perturbing $\tilde{x} \leftarrow \mathsf{P}^{D_0,D_1}(x)$. The probability is over the entire experiment from sampling the learning tasks to the randomness of the perturbation algorithm and the classifier.*

**Discussion.** An alternate definition of hard to classify robustly would be the negation of robust classification. That definition takes a following form:

$$\mathbb{P}_{x \leftarrow D_b}[\exists \tilde{x} \in B(x,\varepsilon) \text{ such that, } \mathsf{R}(\tilde{x}) \neq b] \geq 0.5$$

This definition is unsatisfactory because it does not say anything about how difficult it is to find such perturbations. In the event when such examples are not efficiently discoverable, we do not have to worry about these.

In the definitions used, the perturbing adversary is both efficient and universal. Efficiency is a very natural property to have, in that if the adversarial examples are computationally hard to find, then they are less of a concern. The universality property says that there is a single perturbation adversary that succeeds against all efficient classifiers. This is a strong requirement. This makes our robustly hard to learn tasks better: that they have a unique perturbation adversary that is independent of which classification algorithm is used. On the other hand, it makes our converse results constructing one-way functions from hard to learn robust tasks weaker, because they only hold for such robustly hard to learn tasks, with universal perturbation adversaries.

It is possible to have a perturbation adversary $\mathsf{P}$ that is efficient but not universal. The perturbation adversary gets oracle access to the classifier and has to then output a misclassified example. This is a weaker requirement than Theorem 10. We do not know if such a definition also implies cryptography.

## Appendix B. Learning Parity with Noise

### B.1. Assumption Definition and Discussion

Let $\mathbb{Z}^m_{2,\mathsf{Ham}=\mathsf{t}}$ denote vectors in $\mathbb{Z}^m_2$ with Hamming weight exactly $t$. We will consider Hamming weight as our norm in this setting.

**Definition 11 (Learning Parity with Noise Problem (LPN))** *For $n, m, t \in \mathbb{N}$, an LPN sample is obtained by sampling a matrix $\mathbf{A} \leftarrow \mathbb{Z}_2^{n\times m}$, a secret $\boldsymbol{s} \leftarrow \mathbb{Z}_2^n$, and an error vector $\boldsymbol{\epsilon} \in \mathbb{Z}_{2,\mathsf{Ham}=\mathsf{t}}^m$ and outputting $(\mathbf{A}, \boldsymbol{s}^T\mathbf{A} + \boldsymbol{e}^T)$.*

*We say that an algorithm solves $\mathsf{LPN}_{n,m,t}$ if it distinguishes an LPN sample from a random sample distributed as $\mathbb{Z}_2^{n\times m} \times \mathbb{Z}_2^{1\times m}$.*

**Assumption 12 (Learning Parity with Noise Assumption)** *The Learning Parity with Noise (LPN) assumption assumes that for $m = \mathsf{poly}(n)$ and $t = \theta(m/\sqrt{n})$, the LPN samples are indistinguishable from random. That is, for every efficient distinguisher $\mathsf{D}$,*

$$\Big| \mathop{\mathbb{P}}_{\substack{\boldsymbol{s}\leftarrow\mathbb{Z}_2^n \\ \boldsymbol{e}\leftarrow\mathbb{Z}_{2,\mathsf{Ham}=t}^m}} \left[\mathsf{D}(\mathbf{A}, \boldsymbol{s}^T\mathbf{A} + \boldsymbol{e}^T) = 1\right] - \mathop{\mathbb{P}}_{\boldsymbol{r}\leftarrow\mathbb{Z}_2^m}[\mathsf{D}(\mathbf{A}, \boldsymbol{r}) = 1]\Big| < \mathsf{negl}(n)$$

This regime of parameters $m = \mathsf{poly}(n)$ and $t = \theta(m/\sqrt{n})$ is what is traditionally used to construct public key encryption from the LPN assumption. Next, we consider the LPN problem with preprocessing: in this variant of the problem, an inefficient algorithm $\mathsf{Preprocess}$ is allowed to process the matrix $\mathbf{A}$ arbitrarily to construct a "trapdoor". Then the distinguisher is asked to distinguish LPN sample $(\mathbf{A}, \boldsymbol{s}^T\mathbf{A} + \boldsymbol{e})$ from random. The assumption states that this is difficult for higher error rates.

**Definition 13 (LPN with Preprocessing Problem (LPNP))** *We say that a pair of algorithms $(\mathsf{Preprocess}, \mathsf{D})$ where $\mathsf{Preprocess}$ is possibly inefficient and $\mathsf{D}$ is efficient, solves $\mathsf{LPN}_{n,m,t}$ if $\mathsf{D}$ can distinguish an LPN sample from a random sample given the trapdoor $\mathsf{trap}$ generated by $\mathsf{Preprocess}(\mathbf{A})$.*

The Learning Parity with Noise problem is hard even with preprocessing in the constant noise regime.

**Assumption 14 (LPN with Preprocessing (LPNP))** *Let $m = \mathsf{poly}(n)$ and $t = r \cdot m$ for any constant $r$. For every pair of algorithms $(\mathsf{Preprocess}, \mathsf{D})$ with a possibly inefficient algorithm $\mathsf{Preprocess}$ and efficient $\mathsf{D}$, the following experiment is performed: Sample $\mathbf{A} \leftarrow \mathbb{Z}_2^{n\times m}$ and get $\mathsf{trap} \leftarrow \mathsf{Preprocess}(\mathbf{A})$. Then, the distinguisher $\mathsf{D}$ given $\mathsf{trap}$ cannot distinguish the LPN samples from random. That is for large enough $n$,*

$$\Big| \mathop{\mathbb{P}}_{\substack{\boldsymbol{s}\leftarrow\mathbb{Z}_2^n \\ \boldsymbol{e}\leftarrow\mathbb{Z}_{2,\mathsf{Ham}=t}^m}} \left[\mathsf{D}(\mathsf{trap}, \mathbf{A}, \boldsymbol{s}^T\mathbf{A} + \boldsymbol{e}^T) = 1\right] - \mathop{\mathbb{P}}_{\boldsymbol{r}\leftarrow\mathbb{Z}_2^m}[\mathsf{D}(\mathsf{trap}, \mathbf{A}, \boldsymbol{r}) = 1]\Big| < \mathsf{negl}(n)$$

*where the probability is over the code $\mathbf{A}$, $\boldsymbol{s}, \boldsymbol{e}, \boldsymbol{r}$ and the randomness of the distinguisher $\mathsf{D}$.*

**Discussion.** The most important parameter of the LPN problem is its error rate, that is $r = t/m$. The higher the error rate, the more difficult the problem. There are two important regimes of the error rate: $r$ is a constant and $r = o(\frac{1}{\sqrt{n}})$. When the error rate is a constant, the hardness of LPN in this regime implies one-way functions and hence symmetric key cryptography. We do not know how to base public key encryption on error rates in this regime. When the error rate decreases below $O(\frac{1}{\sqrt{n}})$, we can construct public key encryption from this problem. For error rates below $\log n/n$, the problem becomes easy. The best known algorithms for solving LPN are due to Blum Kalai and Wasserman (Blum et al., 2003) which solves LPN in time $2^{O(n/\log n)}$ requiring $2^{O(n/\log n)}$

samples; and Lyubashevsky (Lyubashevsky, 2005) which solves LPN in time $2^{O(n/\log\log n)}$ with polynomially many samples. For structured LPN samples, more efficient algorithms are known Arora and Ge (2011). Our error distributions are not structured.

Note that the lesser used variant of LPN is used here, in that we insist that the Hamming weight of the error vector is exactly $t$ instead of a random variable. This is equivalent to the standard formulation (Jain et al., 2012).[6] This is done for convenience and the example can be translated to the definition of LPN where the error vector is drawn from a product distribution.

We also consider a the preprocessing variant of Learning Parity with Noise. In this variant, the adversary is allowed to preprocess the code and generate a small "trapdoor" to the code. Then an efficient adversary is tasked with distinguishing the LPN samples from random. The preprocessing variant of LPN assumption states that even this is hard in the constant error regime, that is when $t/m$ is a constant. It is known that decoding linear codes is NP-hard in the worst case (Bruck and Naor, 1990). The search analog of LPN is precisely the average-case variant of this question and is conjectured to be hard in the regime of constant noise rate.

**Trapdoor for Efficient Decoding.** In the public key regime, we want to show that trapdoors exist that enable effient distinguishing of LPN samples. We state the result next: that there is a way to sample a random matrix $\mathbf{H}$ that is indistinguishable from a random matrix such that it has a trapdoor that enables efficient distinguishing.

**Lemma 15 (Computational Trapdoor Sampling)** *Consider the following algorithm* LPNTrapSamp *such that,* LPNTrapSamp *on input* $(n,t)$ *with* $t=\theta(\sqrt{n})$ *does the following:*

LPNTrapSamp$(n,t)$*:*

1. *Sample* $\mathbf{A}\leftarrow\mathbb{Z}_2^{n\times m}$, $\mathbf{S}\leftarrow\mathbb{Z}_2^{n\times n}$ *and* $\mathbf{E}\in\mathbb{Z}_2^{n\times m}$ *where* $\boldsymbol{e}_{(i,\cdot)}\leftarrow\mathbb{Z}_{2,\mathsf{Ham}=t}^{m}$.
2. *Output* $\mathbf{H}=\begin{bmatrix}\mathbf{A}\\ \mathbf{SA}+\mathbf{E}\end{bmatrix},\mathbf{E}$

*The algorithm has the following properties:*

1. $\mathrm{rowspan}(\mathbf{E})\subset\mathrm{rowspan}(\mathbf{H})$ *where* $\mathrm{rowspan}(\mathbf{T})=\{\boldsymbol{s}\mathbf{T}:\boldsymbol{s}\in\mathbb{Z}_2^n\}$.
2. *The matrix* $\mathbf{H}$ *is computationally indistinguishable from uniformly random matries. That is,*

$$\{\mathbf{H}\leftarrow\mathsf{LPNTrapSamp}(n,t)\}\approx_c\left\{\mathbf{U}\leftarrow\mathbb{Z}_2^{2n\times 8n}\right\}$$

3. *With overwhelming probability over the randomness of the algorithm, it outputs* $\mathbf{E}$ *such that every column of* $\mathbf{E}$ *has Hamming weight at most* $t$ *and every row of* $\mathbf{E}$ *has Hamming weight exactly* $t$*.*

The notion of Trapdoor sampling is very widely used in the context of learning with errors assumption. A trapdoor sampling algorithm samples along with the public matrix $\mathbf{A}$ which is statistically close to a random matrix (representing the code/lattice), a secret "trapdoor". This trapdoor enables solving the bounded distance decoding problem, that is given a point close to a codeword in the code, finds the close codeword. As we know, without this trapdoor, this problem is conjectured to be hard. But the trapdoor enables solving this problem.

6. In the search version of the problem where the adversary has to find $\boldsymbol{s}$ given $\mathbf{A},\boldsymbol{s}^T\mathbf{A}+\boldsymbol{e}^T$, these two versions are equivalent as $t$ takes polynomially many values, hence we can go over all polynomially-many and try solving each exact version).

We have a computational analog of that property for LPN in the "public-key" regime of parameters. We construct that below. Because $\mathbf{E}$ is a sparse matrix, it can be used to solve the problem of distinguishing LPN samples from random and decoding noisy codewords.

**Proof**

By definition, $\mathrm{rowspan}(\mathbf{E}) \subset \mathrm{rowspan}(\mathbf{H})$ and that each row of $\mathbf{E}$ has Hamming weight exactly $t$. We need to show that $\mathbf{H}$ is indistinguishable from random and that every column of $\mathbf{E}$ has at most $t$ ones. The former follows from the Learning Parity with Noise combined with a hybrid argument and the latter from a Chernoff bound.

**Claim 16** *The output distribution of* $\mathbf{H}$ *is computationally indistinguishable from uniform. That is,*

$$\{\mathbf{H} \leftarrow \mathsf{LPNTrapSamp}(n,t)\} \approx_c \left\{\mathbf{U} \leftarrow \mathbb{Z}_2^{2n\times 8n}\right\}$$

**Proof** Observe that the LPN assumption can be restated as, The LPN assumption assumes that the following two distributions are indistinguishable:

$$\begin{bmatrix}\mathbf{A}\\ \boldsymbol{s}^T\mathbf{A}+\boldsymbol{e}^T\end{bmatrix} \approx_c \left\{\mathbf{U} : \mathbf{U} \leftarrow \mathbb{Z}_2^{(n+1)\times m}\right\}$$

where $\boldsymbol{s} \leftarrow \mathbb{Z}_2^n$, $\mathbf{A} \leftarrow \mathbb{Z}_2^{n\times m}$, $\boldsymbol{e} \in \mathbb{Z}_2^m$ is a random vector of Hamming weight $t$. The claim then follows by applying a hybrid argument to each of the rows of $\mathbf{SA}+\mathbf{E}$ and replacing them by random vectors, by viewing them as $\boldsymbol{s}_{(i,\cdot)}\mathbf{A}+\boldsymbol{e}_{(i,\cdot)}$ where $\boldsymbol{s}_{(i,\cdot)}, \boldsymbol{e}_{(i,\cdot)}$ denote the $i$-th row of matrix $\mathbf{S}$ and $\mathbf{E}$ and using the LPN assumption. ■

**Claim 17** *Let* $\boldsymbol{e}_{(\cdot,j)}$ *denote the* $j$*-th column of matrix* $\mathbf{E}$*. Then,*

$$\mathop{\mathbb{P}}_{\mathbf{E}\leftarrow\mathsf{LPNTrapSamp}(n,t)}\left[\exists j, \textit{ such that, } \|\boldsymbol{e}_{(\cdot,j)}\|_{\mathsf{Ham}} > t\right] < 8n\cdot e^{-\frac{7t}{24}}\ .$$

**Proof** The proof follows from Chernoff bound and a union bound. For any fixed column $j$, each coordinate $e_{i,j}=1$ independently with probability $t/8n$ where the probability is over $i$. Hence, for any column $j$, the expected Hamming weight is $\frac{t}{8n}\cdot n = \frac{t}{8}$. By a Chernoff bound, we can observe the following:

$$\begin{aligned}\mathop{\mathbb{P}}_{\mathbf{E}\leftarrow\mathsf{LPNTrapSamp}(n,t)}\left[\sum_i e_{i,j} > t\right] &= \mathop{\mathbb{P}}_{\mathbf{E}\leftarrow\mathsf{LPNTrapSamp}(n,t)}\left[\sum_i e_{i,j} > (1+7)\cdot\mathbb{E}(\sum_i e_{i,j})\right]\\ &\le e^{-7\cdot\frac{t}{8}\cdot\frac{1}{3}}\end{aligned}$$

where the inequality follows from the Chernoff bound in the following form: Let $X_1, X_2, \ldots X_n$ be independent random variables taking values in $\{0,1\}$. Let $X$ be their sum and $\mu = \mathbb{E}\,X$. For any $\delta \ge 1$,

$$\mathbb{P}[X \ge (1+\delta)\mu] \le e^{-\frac{\delta\mu}{3}}\ .$$

A union bound over all $j$ gives us the required bound. ■

Because $t=\sqrt{n}$, the failure probability is negligible. ■

### B.2. No Efficient Robust Classifier Exists

Next, we describe a learning task where while it is possible to inefficiently perform robust classification, no efficient robust classifier exists.

**Theorem 18** *For an $n$, let $m = 8n, t = 2n-1, \varepsilon = 2n$. Consider the following learning task. Let $\mathbf{A} \leftarrow \mathbb{Z}_2^{m \times n}$. Define $D_0, D_1$ as:*

$$D_0^{(\mathbf{A})} = \left\{ \boldsymbol{s}^T \mathbf{A} : \boldsymbol{s} \leftarrow \{0,1\}^n \right\} \text{ and, } D_1^{(\mathbf{A})} = \left\{ \boldsymbol{s}^T \mathbf{A} + \mathbf{1} : \boldsymbol{s} \leftarrow \{0,1\}^n \right\}.$$

*The learning task has the following properties.*

1. *(Learnability) A classifier to distinguish $D_0$ from $D_1$ can be learned from the samples efficiently. Furthermore, it is easy to learn a generator/ evaluator for these distributions.*
2. *(No Efficient Robust Classifier Exists) There exists a perturbation algorithm $\mathsf{P}$ such that there exists no efficient robust classifier $\mathsf{R}$ such that,*

$$\mathbb{P}\left[\mathsf{R}(\tilde{y}) \in \mathsf{R}^{-1}(b)\right] \geq 0.5 + \mathsf{negl}(n)$$

*where the perturbed sample $\tilde{y}$ is generated by sampling $y \leftarrow D_b$ for a random $b$ and is then perturbing $\tilde{y} \leftarrow \mathsf{P}^{D_0, D_1}(y)$ such that $\|y - \tilde{y}\| \leq \varepsilon$.*

**Proof** Learnability of this task is trivial. Given enough samples, the entire subspace spanned by $\mathbf{A}$ is learned and can be sampled from.

In order to show that no efficient robust classifier exists for $\varepsilon = 2n$, we rely on the difficulty of LPN with Preprocessing (Assumption 14). Consider the following perturbing adversary $\mathsf{P}$:

$$\mathsf{P}(\boldsymbol{y}): \text{ Output } \tilde{\boldsymbol{y}} = \boldsymbol{y} + \boldsymbol{e} \text{ where } \boldsymbol{e} \leftarrow \mathbb{Z}_{2,\mathsf{Ham}=t}^m$$

Consider the following pair of algorithms $\mathsf{Preprocess}, \mathsf{D}$: $\mathsf{Preprocess}(\mathbf{A})$ inefficiently finds the best possible efficient robust classifier $\mathsf{R}$ and returns that as the trapdoor $\mathsf{trap} = \mathsf{R}$. The distinguisher $\mathsf{D}$ simply runs the robust classifier $\mathsf{R}$ and returns the answer. It can do this in polynomial time because $\mathsf{R}$ is also polynomial time computable.

The LPNP assumption implies that for this pair of algorithms $(\mathsf{Preprocess}, \mathsf{D})$, LPN is hard to solve. That is, for $\mathbf{A} \leftarrow \mathbb{Z}_2^{n \times m}$, $\mathsf{R} \leftarrow \mathsf{Preprocess}(\mathbf{A})$,

$$\left| \mathop{\mathbb{P}}_{\substack{\boldsymbol{s} \leftarrow \mathbb{Z}_2^n \\ \boldsymbol{e} \leftarrow \mathbb{Z}_{2,\mathsf{Ham}=t}^m}} \left[\mathsf{R}(\mathbf{A}, \boldsymbol{s}^T \mathbf{A} + \boldsymbol{e}^T) = 1\right] - \mathop{\mathbb{P}}_{\boldsymbol{r} \leftarrow \mathbb{Z}_2^m} [\mathsf{R}(\mathbf{A}, \boldsymbol{r}) = 1] \right| < \mathsf{negl}(n) \tag{1}$$

Now a hybrid argument finishes the proof as the following distributions are computationally indistinguishable for $\mathsf{R}$:

$$(\mathbf{A}, \boldsymbol{s}^T \mathbf{A} + \boldsymbol{e}^T) \approx_c (\mathbf{A}, \boldsymbol{r}) \equiv (\mathbf{A}, \boldsymbol{r} + \mathbf{1}) \approx_c (\mathbf{A}, \boldsymbol{s}^T \mathbf{A} + \boldsymbol{e}^T + \mathbf{1})$$

where the two $\approx_c$ statements follow from Eq. (1) and the $\equiv$ follows from the fact that adding any fixed vector to the uniform distribution still remains uniform.

This completes the argument. ■

### B.3. Efficient Robust Classifier Exists but is Hard to Learn

In this section, we describe a learning task where a robust classifier exists, but it is hard to learn. Consider the following classification task : Given a matrix $\mathbf{H} \in \mathbb{Z}_2^{2n \times 8n}$, define $D_0, D_1$ as:

$$D_0^{(\mathbf{H})} = \left\{ \boldsymbol{y} \in \mathbb{Z}_2^{8n} : \mathbf{H}\boldsymbol{y} = 0 \bmod 2 \right\} \text{ and, } D_1^{(\mathbf{H})} = \left\{ \boldsymbol{y} + \mathbf{1} \in \mathbb{Z}_2^{8n} : \mathbf{H}\boldsymbol{y} = 0 \bmod 2 \right\}$$

where both are uniform distributions on the sets and $\mathbf{1}$ is the all ones vector on $8n$ dimensions.

**Theorem 19** *For an $n$, let $t = 2\lfloor \sqrt{n}/6 \rfloor - 1$, such that $t$ is odd. Consider the following learning task. Let $(\mathbf{H}, \mathbf{E}) \leftarrow \mathsf{LPNTrapSamp}(n, t)$. Given a matrix $\mathbf{H} \in \mathbb{Z}_2^{2n \times 8n}$, define $D_0, D_1$ as:*

$$D_0^{(\mathbf{H})} = \left\{ \boldsymbol{y} \in \mathbb{Z}_2^{8n} : \mathbf{H}\boldsymbol{y} = 0 \bmod 2 \right\} \textit{ and, } D_1^{(\mathbf{H})} = \left\{ \boldsymbol{y} + \mathbf{1} \in \mathbb{Z}_2^{8n} : \mathbf{H}\boldsymbol{y} = 0 \bmod 2 \right\}$$

*The learning task has the following properties.*

1. *(Learnability) A classifier to distinguish $D_0$ from $D_1$ can be learned from the samples efficiently. Furthermore, it is easy to learn a generator/ evaluator for these distributions.*
2. *(Existence of an Efficient Robust Classifier) There exists an efficient robust classifier $\mathsf{R}$ such that,*
$$\mathbb{P}_{y \leftarrow D_b} \left[ B(y, \varepsilon) \in \mathsf{R}^{-1}(b) \right] \geq 0.99$$
*where $\varepsilon = \lfloor \sqrt{n} \rfloor$ and $B(y, \varepsilon) = \{y' : \|y - y'\|_{\mathsf{Ham}} \leq \varepsilon\}$.*
3. *(Unlearnability of Robust Classifier) There exists a perturbation algorithm such that no efficiently learned classifier can classify better than chance.*

We drop $\mathbf{H}$ from the notation to avoid clutter and denote the distributions as $D_0, D_1$. Here $\mathbf{H}$ functions as the parity check matrix of the code $D_0$ and $D_1$ is a shift of the code. Observe that Part (1): distinguishing between $D_0$ and $D_1$ is easily done by Gaussian elimination.

We want to show that (2) a robust classifier exists, and, (3) it is difficult to find any robust classifier efficiently. We argue this in the subsequent claims.

**Lemma 20 (Existence of Robust Classifier)** *Consider the following robust classifier:*

$\underline{\mathsf{Robust\ Classifer}\ \mathsf{R}_{\mathbf{E}}(\tilde{\boldsymbol{y}})}$*:*

1. *Compute $\boldsymbol{z} = \mathbf{E}\tilde{\boldsymbol{y}} \bmod 2$.*
2. *If $\|\boldsymbol{z}\|_{\mathsf{Ham}} \leq n$ output $0$ otherwise, output $1$.*

*Then, the following holds:*

$$\mathbb{P}_{\boldsymbol{y} \leftarrow D_b} \left[ \mathsf{R}(\tilde{\boldsymbol{y}}) = b \textit{ for all } \tilde{\boldsymbol{y}} \in B(\boldsymbol{y}, \varepsilon) \right] \geq 0.99$$

*for $\varepsilon = \lfloor \sqrt{n} \rfloor$ and $B(\boldsymbol{y}, \varepsilon) = \{\boldsymbol{y} : \|\tilde{\boldsymbol{y}} - \boldsymbol{y}\|_{\mathsf{Ham}} \leq \varepsilon\}$.*

**Proof**

The correctness of the robust classifier follows from the fact that $\mathbf{E}$ is a sparse matrix where each column has Hamming weight at most $t$. Consider the case when $\boldsymbol{y} \leftarrow D_0$, the other case is analogous. Observe that,

$$\tilde{\boldsymbol{y}} = \boldsymbol{y} + \boldsymbol{\epsilon} \bmod 2$$

where $\|\boldsymbol{\epsilon}\|_{\mathsf{Ham}} \leq \varepsilon \leq \sqrt{n}$. Hence,

$$\mathbf{E}\tilde{\boldsymbol{y}} \bmod 2 = \mathbf{E}(\boldsymbol{y} + \boldsymbol{\epsilon}) = \mathbf{E}\boldsymbol{\epsilon} \pmod 2$$

where the second equality follows from the fact that $\mathbf{H}y = 0 \bmod 2$ and that $\mathrm{rowspan}(E) \subseteq \mathrm{rowspan}(H)$. Observe that each column of $\mathbf{E}$ has at most $t$ ones and that the Hamming weight of $\boldsymbol{\epsilon}$ is at most $\varepsilon$. As, $\mathbf{E}\boldsymbol{\epsilon} = \sum_{j:\epsilon_j=1} \boldsymbol{e}_{(\cdot,j)}$, we can bound the Hamming weight $\|\mathbf{E}\boldsymbol{\epsilon}\|_{\mathsf{Ham}} \leq t\|\boldsymbol{\epsilon}\|_{\mathsf{Ham}} \leq t \cdot \varepsilon \leq n/3$. Hence the classifier would always correctly classify adversarially perturbed samples from $D_0$.

In the other case when $b = 1$ observe that $\mathbf{E} \cdot \mathbf{1} = \mathbf{1}$ because each row of $\mathbf{E}$ has Hamming weight $t$ which is odd. Hence the Hamming weight of $\boldsymbol{z}$ is at least $2n - n/3 > n$ in this case and would be classified correctly. This proves that a robust classifier exists. ■

**Lemma 21 (Hardness of Learning a Robust Classifier)** *There exists a perturbation algorithm* $\mathsf{P}$ *such that for every polynomial time learner* $\mathsf{L}$*, the learner* $\mathsf{L}$ *has no advantage over chance in classifying examples perturbed by* $\mathsf{P}$*. That is,*

$$\mathbb{P}\left[\begin{array}{l} \mathbf{H}, \mathbf{E} \leftarrow \mathsf{LPNTrapSamp}(n,t); \\ \boldsymbol{y} \leftarrow D_b^{(\mathbf{H})} \text{ where } b \leftarrow \{0,1\} \\ \tilde{\boldsymbol{y}} \leftarrow \mathsf{P}^{D_0,D_1}(\boldsymbol{y}); \\ b' \leftarrow \mathsf{L}^{D_0,D_1}(\tilde{\boldsymbol{y}}) \end{array} : b = b'\right] \leq \frac{1}{2} + \mathsf{negl}(n)$$

**Proof** This proof is identical to the proof of security of Aleknovich's public key encryption scheme Alekhnovich (2003).

Observe that $D_0, D_1$ are completely specified by the matrix $\mathbf{H}$. So, the learner gets $\mathbf{H}$ instead of sample access. Consider the following random perturbation algorithm $\mathsf{P}$:

$$\mathsf{P}(\boldsymbol{y}) : \text{Output } \tilde{\boldsymbol{y}} = \boldsymbol{x} + \boldsymbol{\epsilon}, \text{ where } \boldsymbol{\epsilon} \leftarrow \mathbb{Z}_{2,\mathsf{Ham}=\mathsf{t}}^m$$

where $\mathbb{Z}_{2,\mathsf{Ham}=\mathsf{t}}^m$ is the distribution on vectors of Hamming weight $t$. This adversary is adding allowable amount of error as $t < \varepsilon = \sqrt{n}$.

Suppose an efficient learner $\mathsf{L}$ exists that can succeed in this game with high probability, we can break the learning parity with noise assumption. This is done in two steps. In the first step, we replace the parity check matrix $\mathbf{H}$ with a uniformly random matrix $\mathbf{H}'$ this should not noticeably change the success probability because the two distributions are indistinguishable. In the second step, now observe that $\mathbf{H}'$ is a uniformly random parity check matrix hence gives rise to a random code. Now we can apply the LPN assumption again, this time to replace the error $\boldsymbol{\epsilon}$ by a uniformly random vector and not noticably change the success probability. This is a contradiction.

■

## Appendix C. Learning with Errors

### C.1. Preliminaries

In this section, we define the learning with errors problem and describe the notion of trapdoor sampling that it supports. In this section, the norm used is the $\ell_\infty$ norm obtained by embedding $\mathbb{Z}_q$ in $\mathbb{Z}$. That is, for vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{Z}_q^n$, $\|\boldsymbol{x} - \boldsymbol{y}\| = \max_i |x_i - y_i|$ where $|z|$ for $z \in \mathbb{Z}_q$ is obtained by embedding $z \in \{-\lfloor q/2 \rfloor, \ldots, -1, 0, 1, \ldots \lfloor q/2 \rfloor\}$ and taking the absolute value.

**Definition 22 (Learning with Errors Problem)** *For $n, m \in \mathbb{N}$ and modulus $q \geq 1$, distribution for error vectors $\chi \subset \mathbb{Z}_q$, a Learning with Errors (LWE) sample is obtained by sampling $\boldsymbol{s} \leftarrow \mathbb{Z}_q^n$, $\mathbf{A} \leftarrow \mathbb{Z}_q^{n \times m}$, $\boldsymbol{e} \leftarrow \chi^m$ and outputting $(\mathbf{A}, \boldsymbol{s}^T\mathbf{A} + \boldsymbol{e}^T \bmod q)$.*

*We say that an algorithm solves $\mathsf{LWE}_{n,m,q,\chi}$ if it distinguishes LWE sample from a random sample distributed as $\mathbb{Z}_q^{n \times m} \times \mathbb{Z}_q^{1 \times m}$.*

**Assumption 23 (Learning with Errors Assumption)** *The Learning with Errors (LWE) assumption assumes that for $m = \mathsf{poly}(n)$, $q = \Omega(n^3)$ and $\chi$ is truncated discrete gaussian over $\mathbb{Z}_q$ with standard deviation $q/n^2$ truncated to $q/2n$, the LWE samples are indistinguishable from random. That is, for every efficient distinguisher $\mathsf{D}$,*

$$\Big| \mathop{\mathbb{P}}_{\substack{\boldsymbol{s} \leftarrow \mathbb{Z}_2^n \\ \boldsymbol{e} \leftarrow \mathbb{Z}_{2,\mathsf{Ham}=t}^m}} \left[\mathsf{D}(\mathbf{A}, \boldsymbol{s}^T\mathbf{A} + \boldsymbol{e}^T) = 1\right] - \mathop{\mathbb{P}}_{\boldsymbol{r} \leftarrow \mathbb{Z}_2^m} [\mathsf{D}(\mathbf{A}, \boldsymbol{r}) = 1] \Big| < \mathsf{negl}(n)$$

We have written specific versions of the LWE assumption. LWE is conjectured to be hard for a large setting of parameters. For a discussion on parameters, see Peikert (2016).

**Definition 24 (LWE with Preprocessing Problem (LWEP))** *We say that a pair of algorithms $(\mathsf{Preprocess}, \mathsf{D})$ where $\mathsf{Preprocess}$ is possibly inefficient and $\mathsf{D}$ is efficient, solves $\mathsf{LWE}_{n,m,t}$ if $\mathsf{D}$ can distinguish an LWE sample from a random sample given the trapdoor $\mathsf{trap}$ generated by $\mathsf{Preprocess}(\mathbf{A})$.*

The Learning Parity with Noise problem is hard even with preprocessing in the constant noise regime. We state the assumption below formally.

**Assumption 25 (LWE with Preprocessing (LWEP))** *Let $m = n \log q + 2n, q = n^3$ and $\chi$ is a discrete Gaussian with standard deviation $q/100$ truncated to $q/10$. For every pair of algorithms $(\mathsf{Preprocess}, \mathsf{D})$ with a possibly inefficient algorithm $\mathsf{Preprocess}$ and polynomial time $\mathsf{D}$, the following experiment is performed: Sample $\mathbf{A} \leftarrow \mathbb{Z}_2^{n \times m}$ and get $\mathsf{trap} \leftarrow \mathsf{Preprocess}(\mathbf{A})$. Then, the distinguisher $\mathsf{D}$ given $\mathsf{trap}$ cannot distinguish the LPN samples from random. That is,*

$$\Big| \mathop{\mathbb{P}}_{\substack{\boldsymbol{s} \leftarrow \mathbb{Z}_2^n \\ \boldsymbol{e} \leftarrow \mathbb{Z}_{2,\mathsf{Ham}=t}^m}} \left[\mathsf{D}(\mathsf{trap}, \mathbf{A}, \boldsymbol{s}^T\mathbf{A} + \boldsymbol{e}^T) = 1\right] - \mathop{\mathbb{P}}_{\boldsymbol{r} \leftarrow \mathbb{Z}_2^m} [\mathsf{D}(\mathsf{trap}, \mathbf{A}, \boldsymbol{r}) = 1] \Big| < \mathsf{negl}(n)$$

**Definition 26 (Lattice Trapdoor)** *For a matrix $A \in \mathbb{Z}_q^{n \times m}$, we denote by $L^\perp$ the* dual lattice *of $A$ composed of all vectors in the kernel of $A$:*

$$L^\perp = \{x \in \mathbb{Z}^m : Ax = 0 \bmod q\}$$

*A* trapdoor *for $A$ is a short basis for the lattice $L^\perp(A)$.*

In the case of LWE, it is known that we can sample matrices $\mathbf{A}$ from a distribution statistically close to uniformly random along with a trapdoor which allows for efficient distinguishing and recovering the lattice point from a noisy one, for close distances (this is referred to as bounded distance decoding).

**Theorem 27 (Trapdoor Sampling (Gentry et al., 2008))** *There exists an algorithm $\mathsf{TrapSamp}$ such that, $\mathsf{TrapSamp}$ on input $(q, m, n)$ where $m \geq n \log q + 2n$ outputs a pair of matrices $(\mathbf{A}, \mathbf{T})$ where $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, $\mathbf{T} \in \mathbb{Z}_q^{m \times n \log q}$, with the following properties:*

- $\mathbf{AT} = \mathbf{0} \mod q$.
- *The output distribution of* $\mathbf{A}$ *is statistically close to uniform (total variation distance* $< 2^{-O(n)}$*).*
- $\mathbf{T}$ *has only zero-one entries.*

### C.2. No Efficient Robust Classifier Exists

In this section we describe a learning task based on LWE that has no robust classifier. This is identical to the LPN based task except the noise distribution is set differently.

**Theorem 28** *For any* $q = n^3$ *and* $m = n \log q + 2n$*, and* $\chi$ *is a discrete Gaussian with standard deviation* $q/100$ *truncated to* $q/10$*. Consider the following learning task. Let* $\mathbf{A} \leftarrow \mathbb{Z}_q^{m \times n}$*. Define* $D_0, D_1$ *as:*

$$D_0^{(\mathbf{A})} = \left\{ \boldsymbol{s}^T \mathbf{A} : \boldsymbol{s} \leftarrow \{0,1\}^n \right\} \text{ and, } D_1^{(\mathbf{A})} = \left\{ \boldsymbol{s}^T \mathbf{A} + \frac{\boldsymbol{q}}{\mathbf{2}} : \boldsymbol{s} \leftarrow \{0,1\}^n \right\}.$$

*The learning task has the following properties.*

1. *(Learnability) A classifier to distinguish* $D_0$ *from* $D_1$ *can be learned from the samples efficiently. Furthermore, it is easy to learn a generator/ evaluator for these distributions.*
2. *(No Efficient Robust Classifier Exists) There exists a perturbation algorithm* $\mathsf{P}$ *such that there exists no efficient robust classifier* $\mathsf{R}$ *such that,*

$$\mathbb{P}\left[\mathsf{R}(\tilde{y}) \in \mathsf{R}^{-1}(b)\right] \geq 0.5 + \mathsf{negl}(n)$$

*where the perturbed sample* $\tilde{y}$ *is generated by sampling* $x \leftarrow D_b$ *for a random* $b$ *and is then perturbing* $\tilde{x} \leftarrow \mathsf{P}^{D_0, D_1}(x)$ *such that* $\|y - \tilde{y}\| \leq q/10$*.*

The proof is identical to the LPN case, with the perturbation adversary $\mathsf{P}$ instead adding noise distributed according to $\chi^m$.

### C.3. An Efficient Robust Classifier Exists but is Hard to Learn

We define the classification task $(D_0, D_1)$ as follows: Given a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ consider distributions $D_0$ and $D_1$ defined as:

$$D_0^{(\mathbf{A})} = \left\{ \boldsymbol{s}^T \mathbf{A} : \boldsymbol{s} \in \mathbb{Z}_q^n \right\} \text{ and, } D_1^{(\mathbf{A})} = \left\{ \boldsymbol{s}^T \mathbf{A} + \frac{q}{2} \cdot \mathbf{1}^T : \boldsymbol{s} \in \mathbb{Z}_q^n \right\}.$$

where both are uniform distributions on the sets and $\mathbf{1}$ is the all ones vector on $m$ dimensions. We drop $\mathbf{A}$ from the notation to avoid clutter and denote the distributions as $D_0, D_1$.

Hence, the task consists of distinguishing lattice vectors from an affine shift of the lattice. That is, given a vector $x \in (D_0 \cup D_1)$, classify weather $x \in D_0$ or $x \in D_1$. Gaussian elimination accomplishes this task easily. We want to show that (a) a robust classifier exists, and, (b) it is difficult to find any robust classifier efficiently. We argue this based on the learning with errors assumption.

At the heart of the construction is the idea of lattice trapdoors. For a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, the trapdoor is a "short" matrix $\mathbf{T}$ such that $\mathbf{AT} = \mathbf{0} \mod q$. There are two key properties of these

trapdoors that we leverage: (1) This short matrix allows us to solve the "bounded distance decoding (BDD)" problem : that is, given a vector close to the lattice, find the closest lattice vector efficiently. Hence, the trapdoor functions as a robust classifier. Also, we can efficiently sample a random matrix $\mathbf{A}$ together with such a trapdoor. (2) It is hard to find such a trapdoor given the matrix $\mathbf{A}$, even when it exists, because these trapdoors allow us to solve the Learning with Errors problem. This allows us to show that the robust classifier is hard to learn.

**Theorem 29** *For any $q = n^3$ and $m = n \log q + 2n$, consider the following learning task. Let $(\mathbf{A}, \mathbf{T}) \leftarrow \mathsf{TrapSamp}(n, m, q)$. Given a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, define $D_0, D_1$ as:*

$$D_0^{(\mathbf{A})} = \left\{ \boldsymbol{s}^T \mathbf{A} : \boldsymbol{s} \in \mathbb{Z}_q^n \right\} \text{ and, } D_1^{(\mathbf{A})} = \left\{ \boldsymbol{s}^T \mathbf{A} + \frac{q}{2} \cdot \mathbf{1}^T : \boldsymbol{s} \in \mathbb{Z}_q^n \right\}.$$

*The learning task has the following properties.*

1. *(Learnability) A classifier to distinguish $D_0$ from $D_1$ can be learned efficiently.*
2. *(Existence of Robust Classifier) There exists a robust classifier $\mathsf{R}$ such that,*

$$\mathop{\mathbb{P}}_{y \leftarrow D_b} \left[ B(y, q/4m) \in \mathsf{R}^{-1}(b) \right] \geq 0.99$$

   *where $B(y, \varepsilon) = \{y' \in \mathbb{Z}_q^m : \|y - y'\|_\infty \leq \varepsilon\}$.*

3. *(Unlearnability of Robust Classifier) There exists a perturbation algorithm such that no efficiently learned classifier can classify better than chance.*

**Lemma 30 (Existence of a Robust Classifier)** *Consider the following robust classifier $\mathsf{R}$:*

$\underline{\mathsf{Robust\ Classifer}\ \mathsf{R}_{\mathbf{T}}(\tilde{\boldsymbol{y}})}$*:*

1. *Compute $\boldsymbol{z} = \tilde{\boldsymbol{y}}^T \mathbf{T} \bmod q$.*
2. *If $\boldsymbol{z} \in \left\{ \frac{-q}{4}, \ldots, \frac{q}{4} \right\}^n$ output $0$ otherwise, output $1$.*

*Then,*

$$\mathop{\mathbb{P}}_{\boldsymbol{x} \leftarrow D_b} [\mathsf{R}(\tilde{\boldsymbol{x}}) = b \text{ for all } \tilde{\boldsymbol{x}} \in B(\boldsymbol{x}, q/4m)] \geq 0.99$$

**Proof** The correctness of the robust classifier follows from the fact that $\mathbf{T}$ is a zero-one matrix and that the errors are bounded in size. Consider the case when $\boldsymbol{y} \leftarrow D_0$, the other case is analogous. Observe that,

$$\tilde{\boldsymbol{y}} = \boldsymbol{y} + \boldsymbol{e} \bmod q = \boldsymbol{s}^T \mathbf{A} + \boldsymbol{e}^T \bmod q$$

where $\|\boldsymbol{e}\|_\infty \leq \frac{q}{4m}$. Hence,

$$\tilde{\boldsymbol{y}}^T \mathbf{T} \bmod q = (\boldsymbol{s}^T \mathbf{A} + \boldsymbol{e}^T) \mathbf{T} \bmod q = \boldsymbol{e}^T \mathbf{T} \bmod q$$

As $\mathbf{T}$ has only zero-one entries, $\boldsymbol{e}^T \mathbf{T}$ is bounded over integers with the absolute value of each coordinate being at most $m \cdot \|\boldsymbol{e}\|_\infty \leq \frac{q}{4}$. This implies that the robust classifier would correctly output 0 when given perturbed samples from $D_0$. ■

In order to show that it is difficult to recover the robust classifier, we rely on the learning with errors assumption. We consider a perturbation adversary that simply adds random noise to the sample it receives.

**Lemma 31 (Hardness of Learning a Robust Classifier)** *There exists a perturbation algorithm* $\mathsf{P}$ *such that for every polynomial time learner* $\mathsf{L}$*, the learner* $\mathsf{L}$ *has no advantage over chance in classifying examples perturbed by* $\mathsf{P}$*. That is,*

$$\mathbb{P}\begin{bmatrix}\mathbf{A}, \mathbf{T} \leftarrow \mathsf{TrapSamp}(n, m, p); \\ \boldsymbol{x} \leftarrow D_b^{(\mathbf{A})} \text{ where } b \leftarrow \{0,1\} \\ \tilde{\boldsymbol{x}} \leftarrow \mathsf{P}^{D_0, D_1}(x); \\ b' \leftarrow \mathsf{L}^{D_0, D_1}(\tilde{x})\end{bmatrix} : b = b' \leq \frac{1}{2} + \mathsf{negl}(n)$$

**Proof** Observe that $D_0, D_1$ are completely specified by the matrix $\mathbf{A}$ and given $\mathbf{A}$ can be sampled efficiently. So, it suffices to give the learner $\mathbf{A}$ instead of sample access. Consider the following random perturbation algorithm $\mathsf{P}$:

$$\mathsf{P}(\boldsymbol{x}) : \text{Output } \tilde{\boldsymbol{x}} = \boldsymbol{x} + \boldsymbol{e}, \text{ where } \boldsymbol{e} \leftarrow \chi^m.$$

So, the experiment above is equivalent to the following:

$$\mathbb{P}\begin{bmatrix}\mathbf{A}, \mathbf{T} \leftarrow \mathsf{TrapSamp}(n, m, p); \\ \boldsymbol{s} \leftarrow \mathbb{Z}_q^n, \boldsymbol{e} \leftarrow \chi^m, b \leftarrow \{0,1\} \\ b' \leftarrow \mathsf{L}(\mathbf{A}, \boldsymbol{s}^T\mathbf{A} + \boldsymbol{e}^T + b\frac{q}{2} \cdot \mathbf{1}^T)\end{bmatrix} : b = b' \leq \frac{1}{2} + \mathsf{negl}(n)$$

The cruical observation is that the learner's job is to distinguish LWE samples $(\mathbf{A}, \boldsymbol{s}^T\mathbf{A} + \boldsymbol{e}^T)$ from shifted LWE samples $(\mathbf{A}, \boldsymbol{s}^T\mathbf{A} + \boldsymbol{e}^T + \frac{q}{2}\mathbf{1}^T)$. The LWE assumption implies that this is difficult because the two distributions are indistinguishable. That is,

$$(\mathbf{A}, \boldsymbol{s}^T\mathbf{A} + \boldsymbol{e}^T) \approx_c (\mathbf{A}, \boldsymbol{r}^T) \approx_c (\mathbf{A}, \boldsymbol{s}^T\mathbf{A} + \boldsymbol{e}^T + \frac{q}{2} \cdot \mathbf{1}^T)$$

and hence no efficient adversary $\mathsf{L}$ can distinguish between the distribution when $b = 0$ from when $b = 1$. And hence for any efficient adversary, the success probability of classifying these perturbed instances is negligibly close to a half, as desired. ■

Hence, we have described a learning task that is learnable, has a robust classifier, but robust classifiers are computationally hard to learn.

## Appendix D. Using Pseudorandom Functions and Error Correcting Codes

In this section, we formally describe the hard-to-robustly learn task based on one-way functions. There are two main ingredients that we use to construct the learning task: Error Correcting Codes (ECCs) and Pseudorandom Functions (PRFs).

An uniquely decodable binary error correcting code allows encoding messages to redundant codewords such that from any codeword perturbed to some degree, we can recover the encoded message.

**Definition 32 (Uniquely Decodable Error Correcting Code)** *An uniquely decodable binary error correcting code,* $C : \{0,1\}^n \to \{0,1\}^m$ *consists of two efficient algorithms* $\mathsf{Encode}, \mathsf{Decode}$*. The code tolerates error fraction* $e$ *if for all messages* $x \in \{0,1\}^n$*,*

$$\mathsf{Decode}(\tilde{y}) = x \text{ for all } \tilde{y} \in B(\mathsf{Encode}(x), em)$$

*where* $B(\mathsf{Encode}(x), em)$ *denotes the Hamming ball of radius* $em$*.*

We know very good error correcting codes.

**Theorem 33 (Guruswami and Indyk (2001))** *For any constant $\gamma > 0$, there exists a binary error correcting code $C : \{0,1\}^n \to \{0,1\}^m$ where $m = O(n/\gamma^3)$ with a decoding radius of $(\frac{1}{4} - \gamma)m$ with polynomial time encoding and decoding.*

We will use this coding scheme with $\gamma = 1/8$ giving us an error correcting code $C : \{0,1\}^n \to \{0,1\}^m$ where $m = \theta(n)$ and tolerates $m/8$ errors for unique decoding.

A pseudorandom function is a keyed function $F_k : \{0,1\}^{n-1} \to \{0,1\}$ where the secret key is picked uniformly random such that, for every efficient adversary, the output of the function is indistinguishable from the output of a random function. A more formal definition is given below. It is known that pseudorandom functions can be constructed from one-way functions.

**Definition 34 (Goldreich et al. (1986))** *A family of polynomial-time computable functions $\mathcal{F} = \{\mathcal{F}_n\}$ where $\mathcal{F}_n = \{F_k : \{0,1\}^n \to \{0,1\}\}$ where $k \in \{0,1\}^n$ and $n \in \mathbb{N}$ is* pseudorandom *if every polynomial time computable adversary $\mathsf{A}$ cannot distinguish between $\mathcal{F}$ and uniformly random function. That is,*

$$\left| \mathop{\mathbb{P}}_{k \leftarrow \{0,1\}^n} \left[ \mathsf{A}^{F_k}(1^n) = 1 \right] - \mathop{\mathbb{P}}_{U_n \leftarrow \mathcal{U}_n} \left[ \mathsf{A}^{U_n}(1^n) = 1 \right] \right| < \mathsf{negl}(n)$$

*where $\mathcal{U}_n$ is the uniform distribution over all functions from $\{0,1\}^n$ to $\{0,1\}$.*

**Theorem 35 (Goldreich et al. (1986))** *Pseudorandom functions exist if one-way functions exist.*

Next, we informally describe the learning task. Consider the following learning task: The two distributions $D_0, D_1$ are parameterized by the PRF key $k$ and defined as follows:

$$D_0 = (0, \mathsf{Encode}(x, F_k(x))) \text{ and, } D_1 = (1, \mathsf{Encode}(x, 1 - F_k(x))) \,.$$

So, the two distributions are tuples where the first half is which distribution the sample was taken from and the second an error correcting code applied to the tuple $(x, F_k(x) + b)$, that is, either the PRF evaluation at the location $x$ or its complement. Note that without the first bit, classifying the original distributions is computationally infeasible. The pseudorandom function looks random at every new location. Including the bit in the sample itself makes the unperturbed classification task easy. The error correcting code ensures that we have a robust classifier.

**Theorem 36** *Let $\{F_k\}$ for $F_k : \{0,1\}^{n-1} \to \{0,1\}$ be a pseudorandom function family and $C : \{0,1\}^n \to \{0,1\}^m$ where $m = \theta(n)$ be an efficiently decodable error correcting code with decoding algorithm* Decode *that tolerates $m/8$ errors.*

*Consider the following learning task. For a random pseudorandom function key $k$, define:*

$$D_0^{(k)} = \left\{(0, C(x, F_k(x))) : x \leftarrow \{0,1\}^{n-1}\right\} \text{ and, } D_1^{(k)} = \left\{(1, C(x, 1 - F_k(x))) : x \leftarrow \{0,1\}^{n-1}\right\}$$

*supported on $\{0,1\}^m$. The learning task has the following properties.*

1. *(Easy to Learn) A classifier to distinguish $D_0$ from $D_1$ can be learned from the samples efficiently.*

2. *(Robust Classifier Exists) There exists a robust classifier* $\mathsf{R}$ *such that,*

$$\mathop{\mathbb{P}}_{y \leftarrow D_b}\left[B(y, m/8) \in \mathsf{R}^{-1}(b)\right] \geq 0.99$$

*where* $m/8$ *is the decoding radius and* $B(y,d) = \{y' : \|y - y'\|_{\mathsf{Ham}} \leq d\}$.

3. *(A Robust Classifier is hard-to-learn) There exists a perturbation algorithm such that no efficiently learned classifier can classify perturbed adversarial examples better than chance.*

**Proof** To prove Part (1) consider the classifier that outputs the first bit. It works correctly on instances from the distributions. To prove Part (2), we rely on the decoding algorithm. After $d = m/8$ edits to the sample, we can recover the underlying message by ignoring the first bit of the tuple and decoding the rest to get the underlying message of the form $(x, c)$ and then use the PRF to classify. More formally, consider the following robust classifer:

Robust Classifer $\mathsf{R}_k(\tilde{y})$ where $\tilde{y} \in \{0,1\}^{m+1}$:

1. Let $(x, c) = \mathsf{Decode}(\tilde{y}_{2:m+1})$ where $\tilde{y}_{2:m+1}$ are all of $\tilde{y}$ but the first bit.
2. Output 0 if $c = F_k(x)$ else output 1.

Observe that error correcting code ensures that from every perturbed sample, we efficiently recover the encoded message. And then because the message is of the form $(x, F_k(x) + b)$ for class $b$, this allows for correct classification.

To show Part (3), we rely on the unlearnability of the PRF. Consider a perturbing adversary that replaces the first bit of the sample by 0. Classification is now equivalent to predicting $F_k(x)$ given $x$. Because predicting $F_k(x)$ is computationally infeasible to learn, so is a robust classifier. ■

Note that, compared to the previous counter-examples, this example does not rely on public key assumptions. The reason for that is that the samples here are "evasive". In that there is no way to generate fresh samples from the two distributions. So, we cannot translate this to a public key encryption scheme because to encrypt, we need a samples from the distributions $D_0, D_1$ along with the perturbing adversary and we do not have access to these samples.

The hardness of this task comes from the hardness of learning the PRF and not from the perturbations. This is different from the schemes based on LPN and LWE.

## Appendix E. Using Average-Case Hardness and Error Correcting Codes

In this section, we formally state Theorem 1 and provide the proof outlined in Section 3.4. We also give an alternative construction that relies on one-way-permutations, but yields a classification problem with distributions that are efficiently samplable.

We first need the notion of an average-case hard function.

**Definition 37 (Average-Case Hard)** *A boolean function* $g : \{0,1\}^n \to \{0,1\}$ *is* $(s, \delta)$-average-case hard *if for all non-uniform probabilistic algorithms* $A$ *running in time* $s$*,*

$$\mathop{\mathbb{P}}_{A, x \in \{0,1\}^n}[A(x) \neq g(x)] \geq \delta$$

There exists functions $g$ which are $(2^{\Theta(n)}, 1/2 - 2^{-\Theta(n)})$-average-case hard (a random function $g$ will suffice with constant probability).

**Theorem 38** *Let $g : \{0,1\}^n \to \{0,1\}$ be a function that is $(2^{\Theta(n)}, 1/2 - 2^{-\Theta(n)})$-average-case hard, and let $\mathsf{Encode} : \{0,1\}^{n+1} \to \{0,1\}^m$ where $m = \theta(n)$ be an efficiently decodable error correcting code with decoding algorithm $\mathsf{Decode}$ that tolerates $m/8$ errors.*

*Consider the following classification task. Define:*

$$D_0 = \{(0, \mathsf{Encode}(x, g(x))) : x \leftarrow \{0,1\}^n\} \text{ and } D_1 = \{(1, \mathsf{Encode}(x, 1 - g(x))) : x \leftarrow \{0,1\}^n\}$$

*This classification task has the following properties.*

1. *(Easy to Classify) An efficient classifier to distinguish $D_0$ from $D_1$ exists.*
2. *(Robust Classifier Exists) There exists a inefficient robust classifier $\mathsf{R}$ such that,*

$$\mathbb{P}_{y \leftarrow D_b}\left[B(y, m/8) \in \mathsf{R}^{-1}(b)\right] \geq 0.99$$

   *where $m/8$ is the decoding radius and $B(y,d) = \{y' : \|y - y'\|_{\mathsf{Ham}} \leq d\}$.*

3. *(No Efficient Robust Classifier Exists) There exists a perturbation algorithm $\mathsf{P}$ such that there exists no polynomial-time robust classifier $\mathsf{R}$ such that,*

$$\mathbb{P}\left[\mathsf{R}(\tilde{y}) \in \mathsf{R}^{-1}(b)\right] \geq 0.5 + \mathsf{negl}(n)$$

   *where the perturbed sample $\tilde{y}$ is generated by sampling $y \leftarrow D_b$ for a random $b$ and is then perturbing $\tilde{y} \leftarrow \mathsf{P}^{D_0, D_1}(y)$ such that $\|y - \tilde{y}\| \leq \varepsilon$.*

**Proof** This proof closely follows the proof of Theorem 36. For Part (1), the classifier that simply outputs the first bit is always correct. For Part (2), we can robustly classify by using the error correcting code to recover the message $(x, g(x))$ or $(x, 1 - g(x))$, and then we can compute the function $g$ to distinguish between these cases. Specifically, the robust classifier is identical to the one presented in the proof of Theorem 36, but computing the function $g$ instead of $F_k(x)$. For Part (3), we rely on the average-case hardness of $g$. Consider the perturbation adversary that replaces the first bit of the sample by 0. Now, classifying $D_0$ vs $D_1$ with non-negligible advantage is equivalent to predicting $g(x)$ given $x$ with non-negligible advantage. This is impossible in polynomial time by the average-case hardness of $g$, and thus efficient robust classification is impossible. ■

We now describe how to achieve the above properties with distributions that are efficiently samplable. First, recall the notion of a *hard-core bit*: Let $f : \{0,1\}^n \to \{0,1\}^n$ be a one-way function. A predicate $b : \{0,1\}^n \to \{0,1\}$ is a *hard-core bit for $f$* if for all probabilistic polynomial-time algorithms $A$,

$$\mathbb{P}_{x \leftarrow \{0,1\}^n}[A(f(x)) = b(x)] \leq \frac{1}{2} + \mathsf{negl}(n)$$

The construction is as follows.

**Theorem 39** *Let $f : \{0,1\}^n \to \{0,1\}^n$ be a one-way permutation, and let $b : \{0,1\}^n \to \{0,1\}$ be a hard-core bit for $f$. Let $\mathsf{Encode} : \{0,1\}^{n+1} \to \{0,1\}^m$ where $m = \theta(n)$ be an efficiently decodable error correcting code with decoding algorithm $\mathsf{Decode}$ that tolerates $m/8$ errors.*

*Consider the following classification task. Define:*

$$D_0 = \{(0, \mathsf{Encode}(f(x), b(x))) : x \leftarrow \{0,1\}^n\} \text{ and } D_1 = \{(1, \mathsf{Encode}(f(x), 1 - b(x))) : x \leftarrow \{0,1\}^n\}$$

*This classification task has the following properties.*

1. *(Easy to Classify) An efficient classifier to distinguish $D_0$ from $D_1$ exists.*
2. *(Robust Classifier Exists) There exists a inefficient robust classifier* $\mathsf{R}$ *such that,*

$$\mathop{\mathbb{P}}_{y \leftarrow D_b}\left[B(y, m/8) \in \mathsf{R}^{-1}(b)\right] \geq 0.99$$

   *where $m/8$ is the decoding radius and $B(y,d) = \{y' : \|y - y'\|_{\mathsf{Ham}} \leq d\}$.*

3. *(No Efficient Robust Classifier Exists) There exists a perturbation algorithm* $\mathsf{P}$ *such that there exists no polynomial-time robust classifier* $\mathsf{R}$ *such that,*

$$\mathbb{P}\left[\mathsf{R}(\tilde{y}) \in \mathsf{R}^{-1}(b)\right] \geq 0.5 + \mathsf{negl}(n)$$

   *where the perturbed sample $\tilde{y}$ is generated by sampling $y \leftarrow D_b$ for a random $b$ and is then perturbing $\tilde{y} \leftarrow \mathsf{P}^{D_0,D_1}(y)$ such that $\|y - \tilde{y}\| \leq \varepsilon$.*

4. *(Efficiently Samplable) The distributions $D_0, D_1$ can be sampled in polynomial time.*

**Proof** Parts (1)-(3) follow exactly as in the proof of Theorem 38. Note that an inefficent distinguisher can invert $f(x)$ to find $x$, and compute $b(x)$. For Part (4), both distributions are clearly efficiently samplable, by first sampling $x$ and then computing $f(x), b(x)$. ■

## Appendix F. Cryptography from Robustly Hard Tasks

In this section, we show that the existence of tasks with a provable gap in classification and robust classification implies one-way functions and hence a variety of cryptographic primitives that include pseudorandom functions, symmetric key encryption among others.

**Theorem 40** *Provably hard-to-learn robust classifiers imply one-way functions. Given a learning task $D_0, D_1$ such that,*

1. *(Robust Classifier Exists) There exists a robust classifier* $\mathsf{R}$ *such that,*

$$\mathop{\mathbb{P}}_{y \leftarrow D_b}\left[B(y, d) \in \mathsf{R}^{-1}(b)\right] \geq 0.90$$

   *where $d$ is the decoding radius and $B(y,d) = \{y' : \|y - y'\|_{\mathsf{Ham}} \leq d\}$.*

2. *(A Robust Classifier is hard-to-learn) There exists an efficient perturbing adversary* $\mathsf{P}$ *such that every efficiently learned classifier* $\mathsf{L}$ *is not a robust classifier. That is, for a learning task $D_0, D_1 \leftarrow \mathsf{Samp}(\mathsf{n})$ and classifier* $\mathsf{L}$*,*

$$\mathbb{P}\left[\mathsf{L}^{D_0,D_1}(\tilde{x}) = b\right] \leq \frac{1}{2} + 0.1\ .$$

   *where the perturbed sample $\tilde{x} \in B(x,d)$ is generated by sampling $x \leftarrow D_b$ for a random $b \leftarrow \{0,1\}$ and is then perturbing $\tilde{x} \leftarrow \mathsf{P}^{D_0,D_1}(x)$. The probability is over the entire experiment from sampling the learning tasks to the randomness of the perturbation algorithm and the classifier.*

*Then one-way functions exist.*

The proof of this theorem relies on fact that we can construct one-way functions from any two distributions that are staistically far and computationally close. The two distributions considered are the perturbed distributions. That is,

$$D'_0 = \{\mathsf{P}(x) : x \leftarrow D_0\} \text{ and, } D'_1 = \{\mathsf{P}(x) : x \leftarrow D_1\}$$

We show that these two distributions are statistically far and yet computationally indistinguishable giving one-way functions. They are statistically far because the robust classfier can distinguish between them. Hence, the total variation distance between the two has to be large. And that they are computationally close because no efficient algorithm can distinguish between the two. Hence one way functions exist.

**Proof**

We formally state the theorem used below.

**Theorem 41 (Folklore, see e.g., Chap. 3, Ex. 11 Goldreich (2001))** *Given a pair of distributions $(X_0, X_1) \leftarrow \mathcal{F}$ over $\mathcal{X}$ that are statistically far,*

$$d_{TV}(X_0, X_1) = \max_{A:\mathcal{X}\to[0,1]} \mathop{\mathbb{E}}_{x\leftarrow X_0} \mathsf{A}(x) - \mathop{\mathbb{E}}_{x\leftarrow X_1} \mathsf{A}(x) > 0.8$$

*and computationally indistinguishable. That is for every polynomial time adversary $\mathsf{A}$ that gets sample access to the distributions,*

$$\mathop{\mathbb{E}}_{x\leftarrow X_0} \mathsf{A}^{(X_0,X_1)}(x) - \mathop{\mathbb{E}}_{x\leftarrow X_1} \mathsf{A}^{(X_0,X_1)}(x) < 0.4$$

*Then one-way functions exist.*[7]

We want to show that these two distributions are statiscally far and computationally close. This relies on the existence of the robust classifier and the difficultly of learning one respectively.

We start by showing that, $d_{TV}(D'_0, D'_1) \geq 0.8$. To observe this, consider the robust classifier as the distinguisher. This implies that,

$$d_{TV} \geq \mathop{\mathbb{E}}_{x\leftarrow D'_1}[R(x)] - \mathop{\mathbb{E}}_{x\leftarrow D'_0}[R(x)] \geq 0.9 - 0.1 \geq 0.8$$

On the other hand, any efficient distinguisher cannot distinguish between the samples by the assumption. Hence we are done. ■

Another reasonable definition, from which we don't know one-way functions is the following: there exists a perturbation adversary $\mathsf{P}$ that given oracle access to the underlying classifier finds counter examples. That is, $\mathbb{P}_{x\leftarrow D_b}\left[\mathsf{R}(\mathsf{P}^{\mathsf{R},D_0,D_1}(x)) \neq b\right] \geq 0.4$. For this definition, using standard min-max arguments (Impagliazzo, 1995; Freund et al., 1999; Vadhan and Zheng, 2013), we can construct "time-bounded" universal adversaries. That is, for time $T$, there exists a perturbation adversary $\mathsf{P}_T$ running in time $\mathsf{poly}(T)$ that finds adversarial examples for all adversaries running in time $T$ or less. This is insufficient to imply one-way functions though.

7. The constants in the equations are fairly arbitrary. We can replace them by any constants $\alpha, \beta$ where $\alpha^2 > \beta$ and the result holds.

**Public Key Encryption.** The two distributions described above have the following public-key encryption flavor: the robust classifier can serve as the decryption algorithm to distinguish between samples from the perturbed distributions $D'_0, D'_1$. If after seeing enough samples, the learning algorithm can generate fresh samples from the two unperturbed distributions $D_0, D_1$ then we also have an encryption algorithm: to encrypt a bit $b$, first sample from the distribution $D_b$ and run the perturbation adversary $\mathsf{P}$ to generate the encryption of the bit. To decrypt, use the robust classifier.

There are two key ingredients missing: (1) The encryption algorithm $\mathsf{P}$ needs access to fresh samples from the two distributions to encrypt. There are learning tasks where we do not have access to these. (2) The ability to sample the robust classifier along with descriptions of the learning tasks. This might not be feasible, especially when the tasks are not chosen, but supplied by nature.

## Appendix G. A Description of BLPR Example and the Blum-Blum-Shub PRG.

In this section, we describe the BLPR counter-example and the Blum-Blum-Shub pseudorandom generator.

We start by defining the notion of a trapdoor pseudorandom generator. A trapdoor pseudorandom generator $\mathsf{TrapPRG} : \{0,1\}^n \to \{0,1\}^{2n}$ is an expanding function whose outputs are indistinguishable from truly random strings. That is, $\{\mathsf{TrapPRG}(x) : x \leftarrow \{0,1\}^n\} \approx_c \left\{y : y \leftarrow \{0,1\}^{2n}\right\}$. Furthermore, the function has a trapdoor $\mathsf{trap}$ that allows distinguishing the output of the PRG from random outputs. That is, there exists a distinguisher $\mathsf{D}$ that given the trapdoor,

$$\mathop{\mathbb{P}}_{x\leftarrow\{0,1\}^n}[\mathsf{D}(\mathsf{trap}, \mathsf{TrapPRG}(x)) = 1] - \mathop{\mathbb{P}}_{y\leftarrow\{0,1\}^{2n}}[\mathsf{D}(\mathsf{trap}, y) = 1] > 0.99$$

Given a trapdoor PRG, the BLPR learning task $D_0, D_1$ is the following:

$$D_0 = \{(0, \mathsf{TrapPRG}(x)) : x \leftarrow \{0,1\}^n\} \text{ and, } D_1 = \left\{(1, y) : y \leftarrow \{0,1\}^{2n}\right\}.$$

We describe the BBS PRG and its trapdoor property next. The Blum-Blum-Shub pseudorandom generator is defined as follows:

Consider a number $N = pq$ where $p, q$ are primes congruent to $3 \pmod 4$. The seed to the PRG is a random element $x_0 \in \mathbb{Z}_N$. Let $\mathsf{hcb}$ be a hardcore bit[8] of the function $x \to x^2 \pmod N$ (eg parity or the most significant bit).

$\mathsf{BBSPRG}(x_0, m)$:

1. For $i \in [1 : m]$,
   (a) Set $x_i = x_{i-1}^2 \pmod N$.
   (b) Set $y_i = \mathsf{hcb}(x_i)$
2. Output $y_1, y_2, \ldots, y_{m-1}, x_m$.

The trapdoor property BLPR refer to construct the robust classifier is the following one: In the construction of the PRG, the security does not rely on outputting the last entry ($x_m$) in its entireity. Though doing so enables the following "trapdoor" property:

8. A function $\mathsf{hcb}$ is a hardcore bit of a one-way function $f$ has the following property, that if given $y = f(x)$ for a random $x$, $\mathsf{hcb}(x)$ is pseudorandom. That is, given any algorithm that given $y = f(x)$ can predict $\mathsf{hcb}(x)$, then we can use this algorithm to invert $f$ with non-negligible probability.

**Lemma 42** *There exists a distinguisher* $\mathsf{D}$ *that given the factorization of* $N$ *can distinguish between the output of the* BBSPRG *from random strings. That is,*

$$\mathop{\mathbb{P}}_{x_0 \leftarrow \mathbb{Z}_N}[\mathsf{D}_{p,q}(\mathsf{BBSPRG}(\mathsf{x_0}))] - \mathop{\mathbb{P}}_{y \leftarrow \{0,1\}^m}[\mathsf{D}_{p,q}(y)] > 0.99$$

**Proof** [Proof Sketch] The proof relies on the fact that Rabin's one way function $f(x) = x^2 \bmod N$ is a trapdoor function that can be efficiently inverted given the factorization of $N$. Furthermore, the inverse returned is the only square root of $x^2$ that is a square itself. Hence the distinguisher does the following:

$\underline{\mathsf{D}(z)}$:

1. Interpret the input as $y_1, y_2, \ldots y_{m-1}, x_m$.
2. If $x_m$ is not a square mod $N$, output $0$.
3. Compute $x_1, x_2, \ldots x_{m-1}$ as $x_i = f^{-1}(x_{i+1})$.
4. If $y_i = \mathsf{hcb}(x_i)$ for all $i$, return $1$, else return $0$.

Observe that the distinguisher always outputs $1$ on outputs of the PRG. On the other hand, when fed a random string, $x_m$ is not a square with probability $3/4$ and even when it is a square, the probability of each $y_i = \mathsf{hcb}(x_i)$ is exactly $1/2$ independently. Hence the probability that the distinguisher outputs $1$ on a random string is $\frac{1}{4} \cdot (\frac{1}{2})^{m-1}$ which is tiny. ■

Based on this, the BLPR counterexample is the following:

**BLPR Counter-Example** *Let* $N = pq$ *where* $p, q$ *are random* $n$*-bit primes of the form* $3 \pmod 4$. *Let* $m = n^2$. *Define* $D_0, D_1$ *as:*

$$D_0 = \{(0, \mathsf{BBSPRG}(x_0)) : x_0 \leftarrow \mathbb{Z}_N\} \text{ and, } D_1 : \left\{(1, z) : z \leftarrow \{0,1\}^{m+\log N}\right\}$$

*Then, the learning task has the following properties: (1) The distributions are easy to classify non-robustly. (2) There exists an inefficient robust classifier for* $\varepsilon = \theta(\sqrt{n})$. *(3) No efficiently learned classifier can classify better than chance. (4) Given the factorization of* $N$*, there exists an efficient robust classifier for* $\varepsilon = \theta(\sqrt{n})$.

Properties 1, 2, 3 are true. To the best of our knowledge, 4 is not known to be true. As we described earlier, we know of robust classifiers for $\varepsilon = O(1)$. This leaves us with the following open questions.

**Open Questions.**

1. Given factorization of $N$, prove that there exists an efficient robust classifier for $\varepsilon = \omega(1)$-bits.

2. (Perturbation Adversary 1) Consider the perturbation adversary that erases the first bit and adds random noise to each bit of the PRG with prob $1/\sqrt{n}$. Given the factorization a $N$, does there exists an efficient robust classifier for this adversary.

3. (Perturbation Adversary 2) The adversary deletes the last complete entry output by the PRG (i.e., $x_m$). Given the factorization of $N$, can we distinguish this PRG from random, when no other error is added.

Although BBS is a trapdoor PRG, it crucially relies on the fact that $x_m$, the last value is available completely intact. Without access to this value, BBS is still a PRG but it is not clear how to do the trapdoor decoding.

As we described earlier, Open Question 3 is a long-standing open question in the computational number theory community (Heninger, 2019; Green, 2013). And Open Question 1 is a harder variant of that question. Finally, Question 2 asks a error correction or decoding question – given the output of a PRG with random errors, can you recover the original PRG string (even given some trapdoor). We are not aware of any way in which this factorization actually helps decoding under random noise.