How to Sample a Discrete Gaussian (and more) from a Random Oracle

George Lu* Brent Waters†

September 16, 2022

Abstract

The random oracle methodology is central to the design of many practical cryptosystems. A common challenge faced in several systems is the need to have a random oracle that outputs from a structured distribution $\mathcal{D}$, even though most heuristic implementations such as SHA-3 are best suited for outputting bitstrings.

Our work explores the problem of sampling from discrete Gaussian (and related) distributions in a manner that they can be programmed into random oracles. We make the following contributions:

- We provide a definitional framework for our results. We say that a sampling algorithm Sample for a distribution is explainable if there exists an algorithm $\mathsf{Explain}$ which, when given an x in the support of $\mathcal{D}$, outputs an $r \in \{0,1\}^n$ such that $\mathsf{Sample}(r) = x$. Moreover, if x is sampled from $\mathcal{D}$ the explained distribution is statistically close to choosing r uniformly at random. We consider a variant of this definition that allows the statistical closeness to be a "precision parameter" given to the $\mathsf{Explain}$ algorithm. We show that sampling algorithms which satisfy our 'explainability' property can be programmed as a random oracle.
- We provide a simple algorithm for explaining *any* sampling algorithm that works over distributions with polynomial sized ranges. This includes discrete Gaussians with small standard deviations.
- We show how to transform a (not necessarily explainable) sampling algorithm Sample for a distribution into a new Sample' that is explainable. The requirements for doing this is that (1) the probability density function is efficiently computable (2) it is possible to efficiently uniformly sample from all elements that have a probability density above a given threshold p, showing the equivalence of random oracles to these distributions and random oracles to uniform bitstrings. This includes a large class of distributions, including all discrete Gaussians.
- A potential drawback of the previous approach is that the transformation requires an additional computation of the density function. We provide a more customized approach that shows the Miccancio-Walter discrete Gaussian sampler is explainable as is. This suggests that other discrete Gaussian samplers in a similar vein might also be explainable as is.

1 Introduction

The random oracle methodology proposed by Bellare and Rogaway [5] allows one to develop a cryptosystem under the premise that all users have access to an oracle that outputs a random string for each queried input. In practice, when deploying said systems, the calls to the oracle are heuristically replaced with calls to an appropriate hash function, such as SHA-3. While the use of such a heuristic comes with some controversy [13], the methodology has been leveraged for a broad spectrum of problems such as chosen-ciphertext security [17] and non-interactive zero knowledge proofs [16], to name just a few. In addition, it has been key to the development of many practical and deployed cryptosystems.

*University of Texas at Austin. Email: gclu@cs.utexas.edu.

†University of Texas at Austin and NTT Research. Email: bwaters@cs.utexas.edu. Supported by NSF CNS-1908611, CNS-1414082, Packard Foundation Fellowship, and Simons Investigator Award.

Although the heuristic of replacing a random oracle with a hash function such as SHA-3 is naturally aligned with oracles that output random bitstrings, there are many examples of cryptosystems that require random oracles to output from other distributions. For instance, the seminal identity-based encryption (IBE) scheme of Boneh-Franklin [7] uses a random oracle that outputs a bilinear group element, as does the Boneh-Lynn-Shacham signature scheme [8] and the multi-authority Attribute-Based Encryption (ABE) systems of [26]. Other examples include the GPV IBE scheme [19], which needs a random oracle to output a vector over $\mathbb{Z}_p$ for some prime p as well as RSA-based full domain hash signatures [6], which need an element over $\mathbb{Z}_N$ for a composite N. Other works explored constructing random oracles which hash into elliptic curves [20, 23].

If one delves deeper into the deployment of such cryptosystems, we can see that there is no specialized hash function for each of these different domains. Instead, to create a random oracle scheme for a certain distribution $\mathcal{D}$ (e.g. random elements over a particular bilinear group), one will utilize a sampling function Sample. The function Sample will take in a string r and outputs something in the desired domain. The distribution of calling Sample on a random string should be statistically close to that of a given distribution $\mathcal{D}$.

While achieving statistical (or computational) closeness to a given distribution is a necessary property of a sampling function, it is not sufficient, as the sampling function may not allow for the "programmability" of a random oracle necessary in a security proof. For example, suppose we ran the BLS signature scheme over a bilinear group $\mathbb{G}$ of prime order p with generator g, public key g^a and secret key $a \in \mathbb{Z}_p$. In the scheme, a signature on a message m is created as $H(m)^a$ where $H(\cdot)$ is an oracle function that outputs a bilinear group element. Suppose we implement H by employing a random oracle H' that outputs bitstrings alongside a Sample algorithm that computes g^r (interpreting r as an integer) so that $H(m) = g^{H'(m)}$. Such an instantiation will indeed output elements statistically close to random bilinear group elements so long as r is sufficiently long. However, this results in a completely broken cryptosystem. To see this, observe that if an attacker can obtain a signature σ on message m, the attacker can then create a signature $\tilde{\sigma}$ on any other message $\tilde{m}$ by computing $\tilde{\sigma} = \sigma^{H'(\tilde{m})/H'(m)}$. Similar counterexamples exist for the other cryptosystems mentioned.

The more general goal of defining sufficient conditions of replacing cryptographic functionalities with each other has been explored in the line of work on indifferentiability initiated in [28], and further expanded on in papers such as [31]. Indeed, the application of the indifferentiability framework with regard to sampling from particular elliptic curve groups was explored in [11].

We define an important property for any sampler to also have a property we call explainability. This can be viewed as a relaxation of indifferentiability specific to sampling functionalities. Roughly, given an element x in the domain, it should be possible to efficiently "reverse sample" an r such that $\mathsf{Sample}(r) = x$. Moreover, the distribution of receiving an x from $\mathcal{D}$ and then outputting r from reverse sampling should be close to just choosing r uniformly at random. Prior works dealt with this issue with various levels of formality. For many of the aforementioned works, 'ad-hoc' workarounds often invoking specific cryptographic assumptions are used to obtain a proof from the plain random oracle model. In bilinear groups, one often calls this a "hash-to-point" function which is present for many bilinear groups, but not necessarily guaranteed. In general there can exist distributions where sampling cannot be explained; consider if the function Sample were a one way function.

Sampling and Explaining Discrete Gaussian Distributions

In this work we explore the problem of sampling and explaining discrete Gaussian distributions. And consequentially, the problem of programming discrete Gaussian distributions into random oracles. Discrete Gaussian distributions are heavily utilized in the design and analysis of lattice-based cryptosystems, a flourishing area of research over the last several years. The problem of sampling discrete Gaussians has been well studied. While it is possible to sample such distributions using a very basic form of rejection sampling [19], further works have both improved on the efficiency of the rejection sampling method [10, 12, 25], as well as explore other techniques to sample from discrete Gaussians [15, 30, 2, 29, 18, 24, 33, 4, 22, 32] such as computing the cumulative density function, or taking convolutions of smaller standard deviation discrete

Gaussians. The goal of most of these works focus on making these samplers more secure and usable, allowing for features such as being constant time, providing better memory-time tradeoffs, or supporting a greater deal of offline precomputation.

To the best of our knowledge, however, the problem of explaining and programming random oracles with discrete Gaussian distributions has received little attention to date, with a few exceptions. One exception is the universal sampler work of Hofheinz et al. [21] that implicitly shows how indistinguishability obfuscation can be used to obtain a computational form of explainability for any efficiently sampleable distribution from a random oracle. However, all current indistinguishability obfuscation candidates are highly impractical and at best invoke further computational assumptions that go beyond those typically used in lattice-based cryptosytems. Our solutions will be both statistical and significantly more efficient. In a more recent work [1], Agrawal, Wichs and Yamada sketch how the rejection sampling algorithm of Gentry-Peikert and Vaikuntanathan [19] is explainable.

Interestingly, the problem of explaining such distributions has come up in multiple contexts. Brakerski, Cash, Tsabary, and Wee [9] gave a homomorphic ABE scheme provably secure from the LWE assumption in the random oracle model. At one point, their construction required a random oracle that outputs a discrete Gaussian. Since no such solution were available, the authors worked around this by using a specialized sampler due to Lyubashevsky and Wichs [27] which is a blend of a standard discrete Gaussian and a binary string. Another example is in a recent multi-authority scheme of Datta, Komargodski and Waters [14] where the need for a random oracle that outputs discrete Gaussians arises. In this case the authors compensate by using a random oracle that outputs an integer over a subexponentially large integer range that can hide a smaller discrete Gaussian by smudging [3]. In this case the workaround resulted in a subexponentially large modulus. (It should be noted that smudging was used elsewhere in their analysis as well).

Finally, we want to emphasize that the need to explainably sample distributions also arises outside of the random oracle model. An interesting example comes up in the aforementioned work of [1], which requires public parameters that can generate discrete Gaussians, but should look like uniform bitstrings. To prove security the authors require the distribution to be explainable.

We advocate for this importance of studying the problem of explaining and programming discrete Gaussian distributions. Ideally, such solutions will match the performance of the prior works on discrete Gaussian sampling (e.g. [15, 30, 2, 29, 18, 24, 33, 4, 22, 32]) that were focused on performance, but not explainability. Pursuing this goal is a natural and fundamental property given the important role of discrete Gaussians in lattice-based cryptography.

1.1 Our Contributions

Our work consists of the following contributions.

Definitional:

We begin by providing a definitional framework for describing our results. We define an explainable sampling system for distribution $\{\mathcal{D}_\lambda\}_\lambda$ to have two efficiently computable algorithms. The first is a $\mathsf{Sample}(1^\lambda; r)$ algorithm which is parameterized by a security parameter and takes in random coins r. This algorithm should output a distribution statistically close (in λ) to $\mathcal{D}_\lambda$ when r is chosen randomly. The second algorithm is a randomized algorithm $\mathsf{Explain}(1^\lambda, 1^\kappa, x)$ that takes as input the security parameter, a "precision" parameters κ and an element $x \in \mathcal{D}_\lambda$. Its job is to output an r such that $\mathsf{Sample}(1^\lambda; r) = x$. In addition, calling $\mathsf{Explain}$ on an x sampled from $\mathcal{D}$ should have a statistically close distribution to that of simply choosing r at random.

A particular feature of our definition is the use of a tunable precision parameter κ where we only require a statistical distance of $\frac{1}{\kappa}$, as opposed to requiring the statistical distance (or computational advantage) to be negligibly close like in indifferentiability. We show that this is sufficient to allow for proofs to go through in programming a random oracle. Intuitively, the parameter κ will in a reduction will be tuned to an attacker's advantage. We note that the parameter κ is only used in the $\mathsf{Explain}$ algorithm and is not a priori set in the Sample algorithm used in a construction. Thus it can be adjusted to fit a particular attacker with a particular advantage in a reduction. This relaxation allows us to prove explainability, and hence show equivalence of random oracles, to a broader range of distributions than otherwise.

We note that our use of a tunable precision parameter is the main definitional difference between our framework and other works that explored reverse sampling.

Sampling over Small Ranges:

We next show a simple algorithm for explainable sampling over small ranges. Suppose that $\mathbb{R}_\lambda$ is the range of distribution $\mathcal{D}_\lambda$ where $|\mathbb{R}_\lambda|$ grows polynomially in λ. We show that *any* sampling algorithm Sample for $\{\mathcal{D}_\lambda\}_\lambda$ is explainable. One simply calls Sample repeatedly until it outputs a desired x or until $|\mathbb{R}_\lambda| \cdot \kappa$ attempts occur without success. This simple process illustrates the flexibility given by the precision parameter in our framework.

A basic corollary extends this lemma to any distribution which is statistically close to a distribution with polynomial support. This implies sampling for discrete Gaussians with poly-sized standard deviations.

Generic Sampling over Conforming Distributions:

A drawback of the previous approach is that it is only applicable when the distribution range is small, and thus cannot be used to explain, say, a discrete Gaussian with a super-polynomial sized standard deviation. We show an approach to generically sample from a broad class of distributions that includes discrete Gaussians with large standard deviations. Unlike the prior solution we will not be able to use a Sample algorithm as is and explain it. Instead we will transform it into a new Sample' algorithm that is explainable. For our transformation we will require a distribution with:

1. A (not necessarily explainable) Sample algorithm for the distribution.
2. A probability density function $\mathsf{ProbDens}$ where $\mathsf{ProbDens}(1^\lambda, x)$ returns a value proportional to the probability x occurs for distribution $\mathcal{D}_\lambda$.
3. A "heavy element sampler" $\mathsf{SampleUniform}$ where $\mathsf{SampleUniform}(1^\lambda, p; r)$ is an explainable sampling algorithm that samples uniformly from all elements in the range of $\mathcal{D}_\lambda$ that have a probability density above p.

Given these we can build an explainable Sample' algorithm that operates by first sampling x' from Sample. Then computing the probability density p' of x'. Next, it randomly "scales" p' by choosing a random $s_0 \in [0, 1]$ and letting $p = s_0 p'$. Finally, it outputs $x \leftarrow \mathsf{SampleUniform}(1^\lambda, p; r)$. We go on to show that this distribution is explainable.

We additionally give an explicit heavy element sampler for the case of discrete Gaussians, and in general observe that this primitive can be computed from the probability density function for many natural distributions. This gives us an explainable sampler for all discrete Gaussians with exponentially bounded centers and standard deviations.

Finally, we show that something akin to the heavy element sampler is needed for a truly generic transformation. We describe oracles which describe a distribution which has the first two properties above, but not the third and show that it is impossible to create an explainable sampler.

Explaining the Miccancio-Walter '17[29] sampling algorithm:

While the previous technique can create a sampler for discrete Gaussians with large standard deviations, it requires creating a new sampler rather than using one as is. If such a sampler is in a critical path, the additional introduced overhead of a high precision computation of the probability distribution function may be undesirable.

As our final contribution, we provide a tailored explain algorithm to the MW sampler. With any sampler, proving its explainability is vital to enabling its use in random oracle based applications, and allows us to securely instantiate such cryptosystems while carrying over the performance benefits of the sampler in question. While we focus on the MW sampler in particular, we believe that the ideas we demonstrate can extend to similar works [30] and provide techniques for showing explainability of other classes of discrete Gaussian samplers.

2 Preliminaries

We say a function $\mathsf{negl}(x)$ is negligible if for all polynomials $p(x)$, there exists some N such that $\forall x > N, \mathsf{negl}(x) < \frac{1}{p(x)}$. A function is noticeable if it is not negligible. The notation $\mathsf{poly}(x)$ will be used to refer to a polynomial function in x, and $\mathsf{EXP}(x)$ will refer to a function $\leq 2^{\mathsf{poly}(x)}$. We will indicate sampling an element from a probability distribution $\mathcal{D}$ as $x \xleftarrow{R} \mathcal{D}$. Similarly, we will use $x \xleftarrow{R} S$ to indicate sampling over the uniform distribution on a set S. We will use $[a, b]$ and (a, b) to denote the closed and open interval from a to b respectively on $\mathbb{R}$. We will subscript the brackets with $\mathbb{Z}$ to denote the same interval on $\mathbb{Z}$. We will use $\text{range}(\mathcal{D})$ to refer to the set of elements in $\mathcal{D}$ which occur with probability > 0.

We will use log as logarithms base 2 if no base is explicitly specified. We use $\lfloor x \rfloor$, $\lceil x \rceil$ to refer to the usual floor and ceiling rounding operations to the integers, and $\lfloor x \rceil$ to mean a randomized rounding to $\lceil x \rceil$ with probability $x \mod 1$ and otherwise $\lfloor x \rfloor$. We subscript rounding operations (e.g. $\lfloor x \rfloor_k$) to round to $\mathbb{Z}/2^k$ instead of $\mathbb{Z}$.

We use statistical distance between two probability distributions $\mathcal{D}_1$ and $\mathcal{D}_2$ to refer to

$$\max_{A \subseteq \text{range}(\mathcal{D}_1) \cup \text{range}(\mathcal{D}_2)} \left| \Pr[x_1 \xleftarrow{R} \mathcal{D}_1, x_1 \in A] - \Pr[x_2 \xleftarrow{R} \mathcal{D}_2, x_2 \in A] \right|$$

We will denote a family of distributions indexed by λ as $\{\mathcal{D}_\lambda\}_\lambda$. We say two distribution families $\{\mathcal{D}^1_\lambda\}_\lambda$, $\{\mathcal{D}^2_\lambda\}_\lambda$ are statistically close if exists some negligible function $\mathsf{negl}(\lambda)$ such that the statistical distance between $\mathcal{D}^1_\lambda$ and $\mathcal{D}^2_\lambda$ is $< \mathsf{negl}(\lambda)$.

We notate a randomized function f which takes as input x and randomness r by $f(x; r)$. For brevity, the function may be written as $f(x)$ when fresh randomness r is used but does not need to be referenced.

Definition 1. *We say a function* $\mathsf{ProbDens}(1^\lambda, x)$ *with domain* $x \in D_\lambda$ *computes the probability density of a distribution family* $\{\mathcal{D}_\lambda\}_\lambda$ *if it runs in* $\mathsf{poly}(\lambda)$ *time, returns a nonnegative integer, and the distribution where element* x' *is selected with probability*

$$\frac{\mathsf{ProbDens}(1^\lambda, x')}{\sum_{x \in D_\lambda} \mathsf{ProbDens}(1^\lambda, x)}$$

is statistically close to $\mathcal{D}_\lambda$[1].

Since $\mathsf{ProbDens}$ *runs in* $\mathsf{poly}(\lambda)$*, it's output length is at most polynomial, so we can bound the maximum value with some* $\mathsf{PDF}_{\max} = \mathsf{PDF}_{\max}(\lambda) \leq \mathsf{EXP}(\lambda)$.[2]

The discrete Gaussian distribution on a set $S \subseteq \mathbb{R}$ with center c and standard deviation σ is defined as the distribution where an element $i \in S$ is picked with probability

$$\frac{e^{-\frac{(i-c)^2}{2\sigma^2}}}{\sum_{x \in S} e^{-\frac{(x-c)^2}{2\sigma^2}}}$$

When the set S is not specified, assume $S = \mathbb{Z}$. This distribution is of particular interest to cryptography due to its presence in the learning with errors assumption and other lattice-based cryptosystems. For simplicity, we will mostly consider univariate discrete gaussians, as, much like their continuous counterparts, discrete gaussians over arbitrary multivariate lattices can be generated via a linear transformation on a set of independent univariate discrete gaussians, where the linear transformation is derived from the covariance of the target discrete gaussian.

[1] While this definition is not the most general for probability distributions over infinite sets, it will suffice for the cases we consider

[2] Note that we only require the probability density to be proportional to the probability of an element being sampled, rather than equal to. As such, any $\mathsf{ProbDens}$ function which outputs fixed precision reals can be converted to one which outputs integers as above by simply multiplying a sufficiently large constant. We choose to define our output to be integers to give a convenient fixed granularity

3 Explainable Sampling

In this section we define our notion of explainable sampling. Intuitively a distribution $\{\mathcal{D}_\lambda\}_\lambda$ can be sampled by a function $\mathsf{Sample}(1^\lambda; r)$ if $\mathsf{Sample}(1^\lambda; r)$ gives a distribution that is statistically close (in λ) to $\{\mathcal{D}_\lambda\}_\lambda$ for when the string r is chosen uniformly at random. We will further say that such a sampling algorithm is explainable if given an element x in the domain of the distribution there is a function $\mathsf{Explain}(1^\lambda, x) \to r'$ that will output an r' such that $\mathsf{Sample}(1^\lambda; r') \to x$. Moreover, the process of picking random coins to sample an element compared to first sampling a random element and then calculating the coins should be statistically close. This final property is what allows one to program in a random oracle.

Below we give our formal definitions of a distribution being sampleable and a Sample algorithm being explainable. Then we sketch how a pair of algorithms meeting this criteria can be used in a cryptographic game to sample from a random oracle.

Definition 2. *We say an algorithm* $\mathsf{Sample}(1^\lambda; r \in \{0,1\}^n)$ *is a sampler for probability distribution family* $\{\mathcal{D}_\lambda\}_\lambda$ *if* Sample *runs in* $\mathsf{poly}(\lambda)$ *time and the output of* $\mathsf{Sample}(1^\lambda; r)$ *is statistically close to* $\{\mathcal{D}_\lambda\}_\lambda$.

Definition 3. *We say a* Sample *algorithm using* $n = n(\lambda)$ *bits of randomness is explainable if there exists a (likely randomized) algorithm* $\mathsf{Explain}(1^\lambda, 1^\kappa, x)$ *such that* $\mathsf{Explain}$ *runs in* $\mathsf{poly}(\lambda, \kappa)$ *time, and there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that the statistical distance between the following two distributions is at most* $\frac{1}{\kappa} + \mathsf{negl}(\lambda)$ [3].

Distribution A

- $r \xleftarrow{R} \{0,1\}^n$.
- $x \leftarrow \mathsf{Sample}(1^\lambda; r)$.
- Return r, x.

Distribution B

- $r' \xleftarrow{R} \{0,1\}^n$
- $x \leftarrow \mathsf{Sample}(1^\lambda; r')$.
- $r \leftarrow \mathsf{Explain}(1^\lambda, 1^\kappa, x)$.
- Return r, x.

We make a few brief remarks on our definition. First, notice that a call to $\mathsf{Explain}(1^\lambda, 1^\kappa, x)$ algorithm is not explicitly required to even return an r such that $\mathsf{Sample}(1^\lambda; r) = x$. However, the definition implies that if it does not do so with sufficiently high probability, it will not meet our requirements.

Next, our $\mathsf{Explain}$ definition takes in a "fidelity" parameter κ in unary. Here we only require that the explain algorithm is within $1/\kappa$ statistical difference in the above game. While also requiring the explain algorithm to run in time polynomial in κ and λ.

Our motivation is to allow for greater flexibility in the case where it might be difficult to design a polytime explain algorithm where the statistical difference is negligibly close, but that there is a natural running time versus precision tradeoff in the explain algorithm. As we will see below, the latter is sufficient for proving security in a game which uses said sampler to programs a random oracle. Suppose there exists an attacker that wins with non-negligible probability ϵ in a cryptographic game that samples using a random oracle. In proving security we will "tune" κ, so we can switch from sampling from the random oracle to 'reverse sampling' using explain such that the statistical distance between these two games is still some nonnegligible fraction (say $\epsilon/2$). We again remark that this relaxation to a tunable precision parameter is the main definitional difference between our framework and other works that explored reverse sampling.

Finally, we note that it will often be convenient to interpret the randomness r as being drawn from the uniform distribution on an interval of $\mathbb{Z}$ or an exponentially precise element of $\mathbb{R}$ rather than over uniformly random bits. It is easy to see we can interpret a bitstring as a binary representation either of the aforementioned domains of values, and so we will directly show that r is uniform on said domain rather than on the underlying bit representation.

[3] One could consider a computational analogue of this definition.

3.1 Explainability in Cryptographic Games

We will use $\mathsf{Game}^{\mathsf{R}(\cdot)}(1^\lambda, \mathcal{A})$ to refer to a series of cryptographic game parameterized by λ where parties are permitted oracle access to some $\mathsf{R}(\cdot)$ against an adversary $\mathcal{A}$ consisting of one or more algorithms (also with access to $\mathsf{R}(\cdot)$. We say $\mathsf{Game}^{\mathsf{R}(\cdot)}(1^\lambda, \cdot)$ is secure, if for all adversaries $\mathcal{A}$ which run in $\mathsf{poly}(\lambda)$ time, $\Pr[\mathsf{Game}^{\mathsf{R}(\cdot)}(1^\lambda, \mathcal{A}) = 1] = \mathsf{negl}(\lambda)$. We will refer to the event of Game returning 1 as 'winning'.

Theorem 1. *Suppose* Sample *is an explainable sampler for* $\{\mathcal{D}_\lambda\}_\lambda$ *with corresponding* $\mathsf{Explain}$ *algorithm. Let* $\mathsf{R}(\cdot)$ *be a random oracle to distribution* $\{\mathcal{D}_\lambda\}_\lambda$*, and* $\mathsf{R}'(\cdot)$ *be a random oracle to* (r, x) *where* $r \xleftarrow{R} \{0,1\}^n$ *and* $x = \mathsf{Sample}(1^\lambda; r)$*. Then if* $\mathsf{Game}^{\mathsf{R}}(1^\lambda, \cdot)$ *is secure, then so is* $\mathsf{Game}^{\mathsf{R}'}(1^\lambda, \cdot)$*.*

Proof.

Lemma 1. *Suppose* Sample *is an explainable sampler for* $\{\mathcal{D}_\lambda\}_\lambda$ *with corresponding* $\mathsf{Explain}$ *algorithm. Let* $\mathsf{R}(\cdot)$ *be a random oracle to distribution* $\{\mathcal{D}_\lambda\}_\lambda$*, and* $\mathsf{R}_\kappa(\cdot)$ *be a random oracle to* (r, x) *where* $x \xleftarrow{R} \{\mathcal{D}_\lambda\}_\lambda$ *and* $r \leftarrow \mathsf{Explain}(1^\lambda, 1^\kappa, x)$*. Then if* $\mathsf{Game}^{\mathsf{R}}(1^\lambda, \cdot)$ *is secure, then* $\mathsf{Game}^{\mathsf{R}_\kappa}(1^\lambda, \cdot)$ *is secure for all* $\kappa \in \mathsf{poly}(\lambda)$*.*

Proof. Assume there is some PPT adversary $\mathcal{A}^{\mathsf{R}_\kappa}$ for which there exists $\kappa'(\lambda)$ such that $\mathcal{A}$ wins $\mathsf{Game}^{\mathsf{R}_\kappa}(1^\lambda, \cdot)$ with noticeable probability. Then define $\mathcal{A}'$ to be an adversary for $\mathsf{Game}^{\mathsf{R}}(1^\lambda, \cdot)$ which runs $\mathcal{A}^{\mathsf{R}_\kappa}$ and simulates oracle calls to R_κ by taking calling oracle R and running $\mathsf{Explain}(1^\lambda, 1^\kappa, \cdot)$ on the output. Since this is exactly the same game, we conclude $\mathcal{A}'$ has a noticeable probability of winning $\mathsf{Game}^{\mathsf{R}}(1^\lambda, \cdot)$. Since $\kappa \in \mathsf{poly}(\lambda)$, this is efficient. □

Lemma 2. *Let* $\mathsf{R}_\kappa(\cdot)$ *be a random oracle to* (r, x) *where* $x \xleftarrow{R} \{\mathcal{D}_\lambda\}_\lambda$ *and* $r \leftarrow \mathsf{Explain}(1^\lambda, 1^\kappa, x)$*. and* $\mathsf{R}'(\cdot)$ *be a random oracle to* (r, x) *where* $r \xleftarrow{R} \{0,1\}^n$ *and* $x = \mathsf{Sample}(1^\lambda; r)$ *. Suppose* $\mathsf{Game}^{\mathsf{R}_\kappa}(1^\lambda, \cdot)$ *is secure for all* $\kappa \in \mathsf{poly}(\lambda)$*, then* $\mathsf{Game}^{\mathsf{R}'(\cdot)}(1^\lambda, \cdot)$ *is secure.*

Proof. Again, assume for sake of contradiction there is some adversary PPT $\mathcal{A}^{\mathsf{R}'}$ which wins $\mathsf{Game}^{\mathsf{R}'}(1^\lambda, \cdot)$ with noticeable probability. Specifically, since it is poly time, let's suppose there exists constants a, b such that $\mathcal{A}^{\mathsf{R}'}$ wins $\mathsf{Game}^{\mathsf{R}'}(1^\lambda, \cdot)$ makes at most λ^a queries and wins with probability $> \lambda^{-b}$ infinitely often. However, notice that by Definition 3, the statistical distance between queries to a query to R' and a query to R_κ is $\frac{1}{\kappa}$. Thus, if we set $\kappa = 2\lambda^{a+b}$, we can union bound the total statistical difference of *all* queries with $\frac{1}{2\lambda^b} + \mathsf{negl}(\lambda)$. Observe that this means we can bound

$$\left|\Pr[\mathsf{Game}^{\mathsf{R}'}(1^\lambda, \mathcal{A}) = 1] - \Pr[\mathsf{Game}^{\mathsf{R}_{2\lambda^{a+b}}}(1^\lambda, \mathcal{A}) = 1]\right| \leq \frac{1}{2\lambda^b} + \mathsf{negl}(\lambda)$$

However, Since $\mathcal{A}$ wins $\mathsf{Game}^{\mathsf{R}_{2\lambda^{a+b}}}(1^\lambda, \mathcal{A})$ with probability $\frac{1}{\lambda^b}$ infinitely often, that means it wins $\mathsf{Game}^{\mathsf{R}_{2\lambda^{a+b}}}(1^\lambda, \mathcal{A})$ with probability $\geq \frac{1}{2\lambda^b}$ infinitely often, contradicting the assumption that $\mathsf{Game}^{\mathsf{R}_\kappa}(1^\lambda, \cdot)$ is secure for all $\kappa \in \mathsf{poly}(\lambda)$. □

Taking Lemma 1 and Lemma 2 together gives us the theorem statement.

□

4 Explaining Sampling over Small Ranges with respect to Discrete Gaussian Samplers

We begin by showing that *any* efficient Sample algorithm over a polynomial sized range, $\{R_\lambda\}_\lambda$, is explainable. The core idea is rather simple. A call to $\mathsf{Explain}(1^\lambda, 1^\kappa, x)$ will simply call the Sample algorithm up to $\kappa \cdot |R_\lambda|$ times until x is output.

We show an immediate corollary to the theorem where if a distribution family has a super-polynomial size range $\{R_\lambda\}_\lambda$, but there exists subsets of the range $\{S_\lambda\}_\lambda$ where $S_\lambda \subseteq R_\lambda$ and $\{S_\lambda\}_\lambda$ are polynomially sized, then our sampling algorithm also works for these distribution. In particular, this covers a discrete Gaussian where the standard deviation σ grows polynomially with λ. We will see that this simple procedure

can serve for explaining the "base case" for the Micciancio-Walter algorithm. Below we formally give our theorem and proof.

Theorem 2. *Let* Sample *be an sampler for some distribution family* $\{\mathcal{D}_\lambda\}_\lambda$ *on range* $\{R_\lambda\}_\lambda$*. If* $|R_\lambda| \leq \mathsf{poly}(\lambda)$*, then* Sample *is explainable.*

Proof. The idea here is to simply brute force the sampler output to find a valid randomness to a given element x. We use the fact that the range is polynomial to bound the amount of probability mass which can be contained by 'infrequent' elements. Consider the algorithm below:

$\mathsf{Explain}(1^\lambda, 1^\kappa, x)$

- Repeat $\kappa \cdot |R_\lambda|$ times
  - Select fresh randomness r
  - $x' \leftarrow \mathsf{Sample}(1^\lambda; r)$
  - If $x' = x$, stop and return r.
- Return $\perp$

Claim 1. Explain *runs in* $\mathsf{poly}(\lambda, \kappa)$ *time*

Proof. First, by the efficiency requirement of Definition 2 $\kappa \cdot |R_\lambda|$, $\mathsf{Sample}(1^\lambda; r)$ has runtime $\mathsf{poly}(\lambda)$. Explain simply calls $\mathsf{Sample}(1^\lambda; r)$ up to $\kappa \cdot |R_\lambda|$ many times (along with some other minor efficient computation). Thus, we can bound the runtime with $\kappa \cdot |R_\lambda| \cdot \mathsf{poly}(\lambda)$. Since $|R_\lambda| \leq \mathsf{poly}(\lambda)$ by assumption, this bounds the runtime with $\kappa \cdot \mathsf{poly}(\lambda) \in \mathsf{poly}(\lambda, \kappa)$. □

Claim 2. Explain *returns* $\perp$ *in Game B with probability* $\leq \frac{1}{\kappa}$

Proof. Observe that over the course of Game B, Sample is called on independent randomness up to $\kappa \cdot |R_\lambda| + 1$ times (once directly from the Game and $\kappa \cdot |R_\lambda|$ times in the execution of Game B). Let us call these outputs x_0 and $x_1, x_2, \ldots, x_{\kappa \cdot |R_\lambda|}$ respectively. Let the set $\mathcal{X} = \{x_i : \forall j \neq i \;\; x_j \neq x_i\}$ By pigeonhole, we know that $|\mathcal{X}| \leq |R_\lambda| - 1$. Thus, since the calls are independent, the probability that the single call directly from Game B (x_0) is in this set $\mathcal{X}$ is $\frac{|\mathcal{X}|}{\kappa |R_\lambda| + 1} \leq \frac{|R_\lambda| - 1}{\kappa |R_\lambda| + 1} < \frac{1}{\kappa}$. On the other hand, if this call is not unique, we can see Explain finds $j > 0 : x_j = x_0$ and so does not return $\perp$. □

Claim 3. *The statistical distance of Game A and Game B in Definition 3 using* Explain *is* $\leq \frac{1}{\kappa}$

Proof. We first note that if $\mathsf{Explain}(1^\lambda, 1^\kappa, x)$ returned $r \neq \perp$, then $x = x' = \mathsf{Sample}(1^\lambda; r)$. Thus, since x is the same function of r in both games, it suffices to show that r in Game B has statistical distance $\leq \frac{1}{\kappa}$ to uniform.

Consider an alternate $\mathsf{Explain}'$ which, rather than sampling x' at most $\kappa \cdot |R_\lambda|$ times, samples x' until it finds an appropriate r. Here, we can compute the probability that any particular r is chosen as the probability the x chosen for Explain is equal to $\mathsf{Sample}(1^\lambda; r)$ multiplied by the probability that r is chosen conditional on $x' \leftarrow \mathsf{Sample}(1^\lambda; r)$. This is equal to

$$\Pr[\mathsf{Sample}(1^\lambda) = x] \cdot \frac{2^{-n}}{\Pr[\mathsf{Sample}(1^\lambda) = x']} = 2^{-n}$$

so r is uniform on $\{0,1\}^n$. Now note that $\mathsf{Explain}'$ only differs from Explain when Explain returns $\perp$ before finding such an r. By Claim 2, this happens with probability $< \frac{1}{\kappa}$, bounding the statistical distance. □

□

Corollary 3. *Let* Sample *be a sampler for some distribution family* $\{\mathcal{D}_\lambda\}_\lambda$ *with range* $\{R_\lambda\}_\lambda$*. If there exists sets* $\{S_\lambda\}_\lambda$ *where* $S_\lambda \subseteq R_\lambda$*,* $|S_\lambda| \leq \mathsf{poly}(\lambda)$*, and*

$$\Pr\left[\begin{matrix} x \xleftarrow{R} \mathcal{D}_\lambda \\ x \notin S_\lambda \end{matrix}\right] < \mathsf{negl}(\lambda),$$

then Sample *is explainable.*

Proof. Let $\{\mathcal{D}'_\lambda\}_\lambda$ be the distribution $\{\mathcal{D}_\lambda\}_\lambda$ conditional on the output being $\in \{S_\lambda\}_\lambda$. It is easy to see that $\{\mathcal{D}'_\lambda\}_\lambda$ satisfies the conditions for Theorem 2, and so any sampler for $\{\mathcal{D}'_\lambda\}_\lambda$ is explainable. Since $\{\mathcal{D}'_\lambda\}_\lambda$ is statistically close to $\{\mathcal{D}_\lambda\}_\lambda$, any sampler for $\{\mathcal{D}_\lambda\}_\lambda$ is a sampler for $\{\mathcal{D}'_\lambda\}_\lambda$, and so is explainable. □

5 Sampling and Explaining Conforming Distributions

The previous section showed how one could explain arbitrary sampler so long as the corresponding distributions grew polynomially in the security parameter. In this section we explore a class of distributions for which we can build explainable samplers. This includes distributions with superpolynomial sized ranges. To perform this we require the distribution family to have:

1. A Sample algorithm for the distribution. (It is not necessarily explainable).
2. A probability density function ProbDens where $\mathsf{ProbDens}(1^\lambda, x)$ returns the probability density function of x for distribution $\mathcal{D}_\lambda$.
3. An "heavy element sampler" algorithm SampleUniform where $\mathsf{SampleUniform}(1^\lambda, p; r)$ is an explainable sampling algorithm that samples uniformly at random from all elements in the range of $\mathcal{D}_\lambda$ that have a probability density function value above p.

We show that if a distribution has all three elements, then there exists another pair of algorithms $\mathsf{Sample}', \mathsf{Explain}'$ that comprise an explainable sampler for the family $\{\mathcal{D}_\lambda\}_\lambda$.

We argue for the utility of this transformation by observing that many natural distributions, including the discrete Gaussian, have an easily computable heavy element sampler. If we consider the discrete Gaussian in particular, then a heavy element sampler simply needs to calculate the values x_0, x_1 such that $\mathsf{ProbDens}(1^\lambda, x_0) = \mathsf{ProbDens}(1^\lambda, x_1) = p$, which can be done by explicitly solving for p using the probability density function of $e^{-\frac{(x-c)^2}{2\sigma^2}} = p$. We can then choose integers uniformly at random in the interval $[x_0, x_1]$.

In fact, in general for monotonic distributions (or distributions which can be partitioned into a polynomial number of monotone segments) on ordered sets, we can easily compute an explainable heavy element sampler by binary searching for the endpoints of the ranges of heavy element with only polynomially many calls to the probability density function, then uniformly sampling on the interval found. As long as the underlying domain has an explainable representation (such as $\mathbb{Z}$ or $\mathbb{R}$), this sampler too is explainable. This condition alone encompasses almost all frequently seen distributions such as (discrete) Gaussians, binomial, geometric, Poisson, etc.

We conclude by arguing that having such a heavy element sampler is necessary for a generic transformation. To do this we provide a distribution for which oracle access to the first two properties is not sufficient to construct an explainable sampler to said distribution.

5.1 Explainable Sampling through heavy element samplers

Definition 4. *We say a function* $\mathsf{SampleUniform}(1^\lambda, p; r)$ *is a heavy element sampler for a distribution family* $\{\mathcal{D}_\lambda\}_\lambda$ *with probability density function* ProbDens *if it runs in* $\mathsf{poly}(\lambda)$ *time, and for all* $p \in [0, \mathsf{PDF}_{\max})$*,* $S_p(1^\lambda; r) = \mathsf{SampleUniform}(1^\lambda, p; r)$ *is a sampler for the uniform distribution on elements* $x \in R_\lambda$ *with* $\mathsf{ProbDens}(1^\lambda, x) \geq p$ *(we will notate said set as* R^p_λ*. We say a heavy element sampler is explainable if there exists a* $\mathsf{poly}(1^\lambda, 1^\kappa)$ *algorithm* $\mathsf{ExplainUniform}(1^\lambda, 1^\kappa, p, x)$ *such that for all* $p = p(\lambda) \in \mathsf{EXP}(\lambda)$*,* $E_p(1^\lambda, 1^\kappa, x) = \mathsf{ExplainUniform}(1^\lambda, 1^\kappa, p, x)$ *is an explainer for* S_p*.*

Theorem 4. *Let* $\mathsf{Sample}, \mathsf{ProbDens}, \mathsf{SampleUniform}$ *be a sampler, probability density function, and explainable heavy element sampler for the probability distribution family* $\{\mathcal{D}_\lambda\}_\lambda$ *on range* $\{R_\lambda\}_\lambda$*. Then there exists a* Sample' *which is an explainable sampler for* $\{\mathcal{D}_\lambda\}_\lambda$*.*

Proof. The underlying idea of the construction of this $\mathsf{Sample}', \mathsf{Explain}'$ is very similar to the generic $\mathsf{Explain}$ algorithm in Theorem 2 - we can use a brute force approach to polynomially approximate $\{\mathcal{D}_\lambda\}_\lambda$. However, because the domain can now be superpolynomial, we can no longer hope polynomially approximate the exact elements of $\{\mathcal{D}_\lambda\}_\lambda$ itself. Instead, we categorize elements into 'buckets' of similar approximate probability density. However, ordinarily, finding an element of a similar probability density would not suffice to satisfy the 'correctness' of an $\mathsf{Explain}$ algorithm. Here, we modify our sampler Sample' so that it uses Sample to produce an initial sample in the range, but this is then 'smudged' to an element of the range with similar probability density which is the actual output of Sample'.

By increasing the precision parameter κ, we decrease the size of each bucket, decreasing the statistical distance of $\mathsf{Explain}'$ but simultaneously increasing the expected number of tries to find an element in the same bucket.

Let Sample' be as follows:

$\mathsf{Sample}'(1^\lambda; r = r_0 \in \{0,1\}^n, s_0 \in [0,1], t_0 \in \{0,1\}^n)$

- $x' \leftarrow \mathsf{Sample}(1^\lambda; r_0)$
- $p' \leftarrow \mathsf{ProbDens}(1^\lambda, x')$
- $p \leftarrow p' \cdot s_0$
- $x \leftarrow \mathsf{SampleUniform}(1^\lambda, p; t_0)$
- Return x

As noted in Section 3, we can interpret a uniform bitstring as the binary expansion of a real number $\in [0,1]$. Since randomness s_0 is only used to compute the probability density on $\mathsf{SampleUniform}$, which is integral, it suffices to use only $\log(\mathsf{PDF}_{\max}) \in \log(\mathsf{EXP}(\lambda)) = \mathsf{poly}(\lambda)$ bits of randomness.

5.1.1 Proof of Sampleability

We first prove that Sample' is a good sampler per Definition 2.

Definition 5. *We define the distribution* $\{\mathsf{PDF}(\mathcal{D}_\lambda)\}_\lambda$ *be defined as the joint distribution on two variables* (a,b) *such that the distribution of* a *is* $\mathcal{D}_\lambda$ *and* b *is uniform from* $[0, \mathsf{ProbDens}(1^\lambda, a))$ *where* $\mathsf{ProbDens}$ *is a probability density function of* $\mathcal{D}_\lambda$*.*

Lemma 3. *The following distributions are statistically close*

Distribution $1 = \mathcal{D}_1$

- $(a,b) \xleftarrow{R} \mathsf{PDF}(\mathcal{D}_\lambda)$
- Output (a,b)

Distribution $2 = \mathcal{D}_2$

- $(a',b) \xleftarrow{R} \mathsf{PDF}(\mathcal{D}_\lambda)$
- $a \xleftarrow{R} \mathsf{SampleUniform}(1^\lambda, b)$
- Output (a,b)

Proof. Consider some fixed (a^*, b^*) in the support of $\mathsf{PDF}(\mathcal{D}_\lambda)$. We can compute the explicit probability this element is picked in distribution 1 as the probability a^* is picked - which is $\frac{\mathsf{ProbDens}(a^*)}{\sum_{a \in R_\lambda} \mathsf{ProbDens}(a)}$, multiplied by the probability that b^* is picked given that a^* - which is $\frac{1}{\mathsf{ProbDens}(a^*)}$, as this is a uniform distribution of size $\mathsf{ProbDens}(a^*)$. Together, we get

$$\Pr[(a^*, b^*) \leftarrow \mathcal{D}_1] = \frac{\mathsf{ProbDens}(a^*)}{\sum_{a \in R_\lambda} \mathsf{ProbDens}(a)} \cdot \frac{1}{\mathsf{ProbDens}(a^*)} = \frac{1}{\sum_{a \in R_\lambda} \mathsf{ProbDens}(a)}$$

In distribution 2, we can write the probability (a^*, b^*) occurs as

$$\Pr[(\_, b^*) \leftarrow \mathsf{PDF}(\mathcal{D}_\lambda)] \cdot \Pr[a^* = \mathsf{SampleUniform}(1^\lambda, b^*) | (\_, b^*) \leftarrow \mathsf{PDF}(\mathcal{D}_\lambda)]$$

Note that since $\mathsf{SampleUniform}$ and $\mathsf{PDF}(\mathcal{D}_\lambda)$ are invoked independently, we can ignore the conditional. Using our analysis from distribution 1, we can compute the first probability as

$$\Pr[(\_, b^*) \leftarrow \mathsf{PDF}(\mathcal{D}_\lambda)] = \sum_{a:\mathsf{ProbDens}(a) > b^*} \Pr[(a, b^*) \leftarrow \mathsf{PDF}(\mathcal{D}_\lambda)] = \sum_{a:\mathsf{ProbDens}(a) > b^*} \frac{1}{\sum_{a \in R_\lambda} \mathsf{ProbDens}(a)}$$

Meanwhile, since by definition $\mathsf{SampleUniform}$ outputs a uniform element of sufficiently high probability density, we have that

$$\Pr[\mathsf{SampleUniform}(1^\lambda, b^*) = a^*] = \frac{1}{|\{a : \mathsf{ProbDens}(a) > b^*\}|}$$

Which brings the total probability of $(a^*, b^*) \leftarrow \mathcal{D}_2$ as

$$\sum_{a:\mathsf{ProbDens}(a) > b^*} \frac{1}{\sum_{a \in R_\lambda} \mathsf{ProbDens}(a)} \cdot \frac{1}{|\{a : \mathsf{ProbDens}(a) > b^*\}|} = \frac{1}{\sum_{a \in R_\lambda} \mathsf{ProbDens}(a)}$$

the same as in $\mathcal{D}_1$ □

Lemma 4. Sample' *is an sampler for* $\{\mathcal{D}_\lambda\}_\lambda$.

Proof.

Claim 4. Sample' *produces a distribution statistically close to* $\{\mathcal{D}_\lambda\}_\lambda$.

Proof. Since Sample is a sampler to $\{\mathcal{D}_\lambda\}_\lambda$, the distribution of (x', p) as defined in Sample' is statistically close to $\{\mathsf{PDF}(\mathcal{D}_\lambda)\}_\lambda$. By Lemma 3, the distribution of (x, p) must also be statistically close to $\{\mathsf{PDF}(\mathcal{D}_\lambda)\}_\lambda$. By Definition 5, the distribution of x must be statistically close to $\{\mathcal{D}_\lambda\}_\lambda$. □

Claim 5. Sample' *runs in* $\mathsf{poly}(\lambda)$ *time.*

Proof. Sample' makes a single call to each of $\mathsf{Sample}, \mathsf{ProbDens}, \mathsf{SampleUniform}$, which, by assumption, are $\mathsf{poly}(\lambda)$ time algorithms along with a single multiplication, and so is also $\mathsf{poly}(\lambda)$ time. □

By Claim 4 and Claim 5, Sample' fulfills the definition of a sampler for $\{\mathcal{D}_\lambda\}_\lambda$

□

5.1.2 Proof of Explainability

We next prove that Sample' is explainable per Definition 3. The general idea here is that because x is chosen randomly using the value p generated by $\mathsf{ProbDens}$ rather than directly from Sample, elements x' with similar $\mathsf{ProbDens}$ values are similarly likely to be explanation for x. We can then use our precision parameter κ to define intervals of $\mathsf{ProbDens}(1^\lambda, x')$ and sample an interval of 'acceptable' p values relative to the correct conditional density. By restricting the number and size of such intervals, we can guarantee that a polynomial number of calls to Sample finds an x' in the chosen interval while still ensuring the p is approximated fairly closely.

Lemma 5. Sample' *is explainable.*

Proof. Consider the following $\mathsf{Explain}'$ algorithm for Sample'.

$\mathsf{ProbIndex}(x, \rho)$

- Let $b = 1 + \frac{1}{\rho}$
- If $\mathsf{ProbDens}(x) = 0$, return $\perp$
- Return $\lfloor \log_b(\mathsf{ProbDens}(x)) \rfloor$

$\mathsf{Explain}'(1^\lambda, 1^\kappa, x)$

- $p \xleftarrow{R} [0, \mathsf{ProbDens}(x)]$
- $x_0 \leftarrow \mathsf{SampleUniform}(1^\lambda, p)$
- Run $9 \ln(\mathsf{PDF}_{\max}) \cdot \kappa^2 \cdot \lambda$ times:
  - Generate fresh randomness r'.
  - $x' \leftarrow \mathsf{Sample}(1^\lambda; r')$.
  - If $\mathsf{ProbIndex}(x_0, 3\kappa) = \mathsf{ProbIndex}(x', 3\kappa)$.
    - Set $r_0 = r'$
    - Set $s_0 = \frac{p}{\mathsf{ProbDens}(x')}$
    - Set $t_0 = \mathsf{ExplainUniform}(1^\lambda, 1^{3\kappa}, p, x)$
    - Return $r = (r_0, s_0, t_0)$
- Return $\perp$.

Claim 6. $\mathsf{Explain}'$ *runs in* $\mathsf{poly}(\lambda, \kappa)$ *time.*

Proof. $\mathsf{Explain}'$ utilizes some efficient (polynomial time) subprocedures in $\mathsf{Sample}, \mathsf{SampleUniform}, \mathsf{ProbDens}$, and $\mathsf{ProbIndex}$. Moreover, since $\mathsf{PDF}_{\max} \in \mathsf{EXP}(\lambda)$, $\log(\mathsf{PDF}_{\max}) \in \mathsf{poly}(\lambda)$, so the algorithm loops only a $\mathsf{poly}(\lambda, \kappa)$ amount of times. □

Claim 7. *The statistical distance of Game A and Game B in Definition 3 using* $\mathsf{Explain}'$ *is* $< \frac{1}{\kappa} + \mathsf{negl}(\lambda)$

Proof. We will proceed with a sequence of games argument, where Game 0 is an execution of Game A using Sample' and Game 11 is an execution of Game B using Sample', $\mathsf{Explain}'$.

Game 0

- $r = (r_0, s_0, t_0) \xleftarrow{R} \{0,1\}^n \times \{0,1\} \times \{0,1\}^n$
- $x \leftarrow \mathsf{Sample}'(x; r)$
- Return r, x.

Game 0 (Sample' **Expanded)**

- $r_0 \xleftarrow{R} \{0,1\}^n$, $s_0 \xleftarrow{R} [0,1]$, $t_0 \xleftarrow{R} \{0,1\}^n$
- $x' \leftarrow \mathsf{Sample}(1^\lambda; r_0)$
- $p' \leftarrow \mathsf{ProbDens}(1^\lambda, x')$
- $p = p' \cdot s_0$
- $x \leftarrow \mathsf{SampleUniform}(1^\lambda, p; t_0)$
- Return r_0, s_0, t_0, x.

Game 1

- ~~$r_0 \xleftarrow{R} \{0,1\}^n$,~~ $s_0 \xleftarrow{R} [0,1]$, $t_0 \xleftarrow{R} \{0,1\}^n$
- $x'' \xleftarrow{R} \mathcal{D}_\lambda$
- Run until break[4]
  - Generate fresh randomness r'
  - $x' \leftarrow \mathsf{Sample}(1^\lambda, r')$
  - If $x' = x''$
    - Set $r_0 = r'$ and break
- $p' \leftarrow \mathsf{ProbDens}(1^\lambda, x'')$
- $p = p' \cdot s_0$
- $x \leftarrow \mathsf{SampleUniform}(1^\lambda, p; t_0)$
- Return r_0, s_0, t_0, x.

Game 2

- ~~$s_0 \xleftarrow{R} [0,1]$,~~ $t_0 \xleftarrow{R} \{0,1\}^n$
- $x'' \xleftarrow{R} \mathcal{D}_\lambda$
- Run until break
  - Generate fresh randomness r'
  - $x' \leftarrow \mathsf{Sample}(1^\lambda, r')$
  - If $x'' = x'$
    - Set $r_0 = r'$ and break
- $p'' \xleftarrow{R} [0, \mathsf{ProbDens}(1^\lambda, x'')]$
- Set $s_0 = \frac{p''}{\mathsf{ProbDens}(1^\lambda, x'')}$
- $x \leftarrow \mathsf{SampleUniform}(1^\lambda, p''; t_0)$
- Return r_0, s_0, t_0, x.

Game 3

- $t_0 \xleftarrow{R} \{0,1\}^n$
- $(x'', p'') \xleftarrow{R} \mathsf{PDF}(\mathcal{D}_\lambda)$
- Run until break
  - Generate fresh randomness r'
  - $x' \leftarrow \mathsf{Sample}(1^\lambda, r')$
  - If $x' = x''$
    - Set $r_0 = r'$ and break
- Set $s_0 = \frac{p''}{\mathsf{ProbDens}(1^\lambda, x_0)}$
- $x \leftarrow \mathsf{SampleUniform}(1^\lambda, p''; t_0)$
- Return r_0, s_0, t_0, x.

[4] this process could potentially take unbounded time, but simply act as 'bridging' steps to make the change in distribution easier to see. The final game will be efficient

Game 4

- ~~$t_0 \xleftarrow{R} \{0,1\}^n$~~
- $(x'', p'') \xleftarrow{R} \mathsf{PDF}(\mathcal{D}_\lambda)$
- Run until break
  - Generate fresh randomness r'
  - $x' \leftarrow \mathsf{Sample}(1^\lambda, r')$
  - If $x' = x''$
    - Set $r_0 = r'$ and break
- Set $s_0 = \frac{p''}{\mathsf{ProbDens}(1^\lambda, x'')}$
- $x \leftarrow \mathsf{SampleUniform}(1^\lambda, p'')$
- Set $t_0 = \mathsf{ExplainUniform}(1^\lambda, 1^{3\kappa}, p'', x)$
- Return r_0, s_0, t_0, x.

Game 5

- $(x'', p'') \xleftarrow{R} \mathsf{PDF}(\mathcal{D}_\lambda)$
- $x \leftarrow \mathsf{SampleUniform}(1^\lambda, p'')$
- Run until break
  - Generate fresh randomness r'
  - $x' \leftarrow \mathsf{Sample}(1^\lambda, r')$
  - If $x' = x''$
    - Set $r_0 = r'$
    - Set $s_0 = \frac{p''}{\mathsf{ProbDens}(1^\lambda, x')}$
    - Set $t_0 = \mathsf{ExplainUniform}(1^\lambda, 1^{3\kappa}, p'', x)$
    - Break
- Return r_0, s_0, t_0, x.

Game 6

- $(x'', p'') \xleftarrow{R} \mathsf{PDF}(\mathcal{D}_\lambda)$
- Set $x_0 = x''$
- $p \xleftarrow{R} [0, \mathsf{ProbDens}(1^\lambda, x_0)]$
- $x \leftarrow \mathsf{SampleUniform}(1^\lambda, p)$
- Run until break
  - Generate fresh randomness r'
  - $x' \leftarrow \mathsf{Sample}(1^\lambda, r')$
  - If $x' = x''$
    - Set $r_0 = r'$
    - Set $s_0 = \frac{p}{\mathsf{ProbDens}(1^\lambda, x')}$
    - Set $t_0 = \mathsf{ExplainUniform}(1^\lambda, 1^{3\kappa}, p, x)$

        * Break
- Return r_0, s_0, t_0, x.

Game 7

- $(x'', p'') \xleftarrow{R} \mathsf{PDF}(\mathcal{D}_\lambda)$
- Sample $x_0 \xleftarrow{R} \mathcal{D}_\lambda$ conditional on $\mathsf{ProbIndex}(x_0, 3\kappa) = \mathsf{ProbIndex}(x'', 3\kappa)$
- $p \xleftarrow{R} [0, \mathsf{ProbDens}(1^\lambda, x_0)]$
- $x \leftarrow \mathsf{SampleUniform}(1^\lambda, p)$
- Run until break
  - Generate fresh randomness r'
  - $x' \leftarrow \mathsf{Sample}(1^\lambda, r')$
  - If $x' = x''$
    * Set $r_0 = r'$
    * Set $s_0 = \frac{p}{\mathsf{ProbDens}(1^\lambda, x')}$
    * Set $t_0 = \mathsf{ExplainUniform}(1^\lambda, 1^{3\kappa}, p, x)$
    * Break
- Return r_0, s_0, t_0, x.

Game 8

- $(x_0, p) \xleftarrow{R} \mathsf{PDF}(\mathcal{D}_\lambda)$
- Sample $x'' \xleftarrow{R} \mathcal{D}_\lambda$ conditional on $\mathsf{ProbIndex}(x_0, 3\kappa) = \mathsf{ProbIndex}(x'', 3\kappa)$
- $x \leftarrow \mathsf{SampleUniform}(1^\lambda, p)$
- Run until break
  - Generate fresh randomness r'
  - $x' \leftarrow \mathsf{Sample}(1^\lambda, r')$
  - If $x' = x''$
    * Set $r_0 = r'$
    * Set $s_0 = \frac{p}{\mathsf{ProbDens}(1^\lambda, x')}$
    * Set $t_0 = \mathsf{ExplainUniform}(1^\lambda, 1^{3\kappa}, p, x)$
    * Break
- Return r_0, s_0, t_0, x.

Game 9

- $(x_0, p) \xleftarrow{R} \mathsf{PDF}(\mathcal{D}_\lambda)$
- $x \leftarrow \mathsf{SampleUniform}(1^\lambda, p)$
- Run $9 \ln(\mathsf{PDF}_{\max}) \cdot \kappa^2 \cdot \lambda$ times
  - Generate fresh randomness r'
  - $x' \leftarrow \mathsf{Sample}(1^\lambda, r')$
  - If $\mathsf{ProbIndex}(x_0, 3\kappa) = \mathsf{ProbIndex}(x', 3\kappa)$

        * Set $r_0 = r'$
        * Set $s_0 = \frac{p}{\mathsf{ProbDens}(1^\lambda, x')}$
        * Set $t_0 = \mathsf{ExplainUniform}(1^\lambda, 1^{3\kappa}, p, x)$
        * Break
- Return r_0, s_0, t_0, x.

Game 10

- $(x, p) \xleftarrow{R} \mathsf{PDF}(\mathcal{D}_\lambda)$
- $x_0 \leftarrow \mathsf{SampleUniform}(1^\lambda, p)$
- Run $9\ln(\mathsf{PDF}_{\max}) \cdot \kappa^2 \cdot \lambda$ times
    - Generate fresh randomness r'
    - $x' \leftarrow \mathsf{Sample}(1^\lambda, r')$
    - If $\mathsf{ProbIndex}(x_0, 3\kappa) = \mathsf{ProbIndex}(x', 3\kappa)$
        * Set $r_0 = r'$
        * Set $s_0 = \frac{p}{\mathsf{ProbDens}(1^\lambda, x')}$
        * Set $t_0 = \mathsf{ExplainUniform}(1^\lambda, 1^{3\kappa}, p, x)$
        * Break
- Return r_0, s_0, t_0, x.

Game 11

- $x \leftarrow \mathsf{Sample}'(1^\lambda)$
- $p \xleftarrow{R} [0, \mathsf{ProbDens}(x)]$
- $x_0 \leftarrow \mathsf{SampleUniform}(1^\lambda, p)$
- Run $9\ln(\mathsf{PDF}_{\max}) \cdot \kappa^2 \cdot \lambda$ times
    - Generate fresh randomness r'
    - $x' \leftarrow \mathsf{Sample}(1^\lambda, r')$
    - If $\mathsf{ProbIndex}(x_0, 3\kappa) = \mathsf{ProbIndex}(x', 3\kappa)$
        * Set $r_0 = r'$
        * Set $s_0 = \frac{p}{\mathsf{ProbDens}(1^\lambda, x')}$
        * Set $t_0 = \mathsf{ExplainUniform}(1^\lambda, 1^{3\kappa}, p, x)$
        * Break
- Return r_0, s_0, t_0, x.

Game 11 (Shortened)

- $x \leftarrow \mathsf{Sample}'(1^\lambda)$
- $r = (r_0, s_0, t_0) \leftarrow \mathsf{Explain}(1^\lambda, 1^\kappa, x)$
- Return r, x.

Claim 8. *The distributions output by Game 0 and Game 1 have statistical distance* $\mathsf{negl}(\lambda)$.

Proof. By Definition 2, the distribution of x' in Game 0 (from Sample) and Game 1 (from $\mathcal{D}_\lambda$) are statistically close. Now since r_0 is uniform, the conditional probability of r_0 on any fixed x' is still uniform on all r_0 such that $\mathsf{Sample}(1^\lambda; r_0) = x'$ in both games. The only other changes are notational between x' and x'', which are equal in this game. □

Claim 9. *The distributions output by Game 1 and Game 2 have statistical distance* 0.

Proof. This game only changes the way s_0 is generated. In Game 1, the distribution of s_0 is uniform on $[0, 1]$. In Game 2, it is the quotient of p'' which is uniform on $[0, \mathsf{ProbDens}(1^\lambda, x'')]$ and $\mathsf{ProbDens}(1^\lambda, x'')$, which is simply uniform on $[0, 1]$. □

Claim 10. *The distributions output by Game 2 and Game 3 have statistical distance* 0.

Proof. By definition, $(x'', p'') \xleftarrow{R} \mathsf{PDF}(\mathcal{D}_\lambda)$ is defined to be $x'' \xleftarrow{R} \mathcal{D}_\lambda$ and $p'' \xleftarrow{R} [0, \mathsf{ProbDens}(1^\lambda, x'')]$, which is exactly how it is generated in Game 3. □

Claim 11. *The distributions output by Game 3 and Game 4 have statistical distance* $\frac{1}{3\kappa} + \mathsf{negl}(\lambda)$.

Proof. By Definition 3, the distributions

Game A	Game B
• $t_0 \xleftarrow{R} \{0,1\}^n$.	• $x \leftarrow \mathsf{SampleUniform}(1^\lambda, p)$.
• $x \leftarrow \mathsf{SampleUniform}(1^\lambda, p; t_0)$.	• $r \leftarrow \mathsf{ExplainUniform}(1^\lambda, 1^{3\kappa}, p, x)$.
• Return t_0, x.	• Return t_0, x.

have statistical distance $\frac{1}{3\kappa} + \mathsf{negl}(\lambda)$. Note this corresponds exactly to how t_0, x are generated in Games 3 and 4 respectively.

□

Claim 12. *The distributions output by Game 4 and Game 5 have statistical distance* 0.

Proof. The changes in Game 5 only change the order some variables are generated, but not the way they are generated. □

Claim 13. *The distributions output by Game 5 and Game 6 have statistical distance* 0.

Proof. In this game, we substitute all uses of p'' with p. By Definition 1, $x'' \xleftarrow{R} \mathcal{D}_\lambda$, so (x'', p) is distributed according to $\mathsf{PDF}(\mathcal{D}_\lambda)$, so are statistically identical. □

Claim 14. *The distributions output by Game 6 and Game 7 have statistical distance* $\frac{1}{3\kappa}$.

Proof. In this game, rather than setting $x_0 = x''$, we set x_0 to be an element with the same $\mathsf{ProbIndex}$ as x''. Observe that x_0 is only used to sample p, and by definition of $\mathsf{ProbIndex}$, $\frac{|\mathsf{ProbDens}(1^\lambda, x_0) - \mathsf{ProbDens}(1^\lambda, x'')|}{\mathsf{ProbDens}(1^\lambda, x'')}$ is at most $\frac{1}{3\kappa}$, and so we can conclude that the uniform distribution on $[0, \mathsf{ProbDens}(1^\lambda, x_0)]$ and $[0, \mathsf{ProbDens}(1^\lambda, x'')]$ have statistical distance at most $\frac{1}{3\kappa}$. □

Claim 15. *The distributions output by Game 7 and Game 8 have statistical distance* 0.

Proof. By this game, note that p'' is unused, so we can examine the difference in how the joint distribution on x_0, x'', p is generated. Let ${x_0}^*, {x''}^*, p^*$ be some set of values taken by x_0, x'', p. We can see the probability density of this is in Game 7 proportional to

$$\Pr_{x'' \xleftarrow{R} \mathcal{D}_\lambda}[{x''}^* = x''] \cdot \Pr_{x_0 \xleftarrow{R} \mathcal{D}_\lambda}[{x_0}^* = x_0 | \mathsf{ProbIndex}({x''}^*, 3\kappa) = \mathsf{ProbIndex}(x_0, 3\kappa)] \cdot \frac{1}{\mathsf{ProbDens}(1^\lambda, x_0)}$$

$$= \Pr_{x'' \xleftarrow{R} \mathcal{D}_\lambda}[{x''}^* = x''] \cdot \frac{\Pr_{x_0 \xleftarrow{R} \mathcal{D}_\lambda}[{x_0}^* = x_0]}{\Pr_{x_0 \xleftarrow{R} \mathcal{D}_\lambda}[\mathsf{ProbIndex}({x''}^*, 3\kappa) = \mathsf{ProbIndex}(x_0, 3\kappa)]} \cdot \frac{1}{\mathsf{ProbDens}(1^\lambda, x_0)}$$

by definition of conditional probability

$$= \frac{\Pr_{x'' \xleftarrow{R} \mathcal{D}_\lambda}[{x''}^* = x'']}{\Pr_{x_0 \xleftarrow{R} \mathcal{D}_\lambda}[\mathsf{ProbIndex}({x''}^*, 3\kappa) = \mathsf{ProbIndex}(x_0, 3\kappa)]} \cdot \Pr_{x_0 \xleftarrow{R} \mathcal{D}_\lambda}[{x_0}^* = x_0] \cdot \frac{1}{\mathsf{ProbDens}(1^\lambda, x_0)}$$

via simple algebraic manipulation

$$= \frac{\Pr_{x'' \xleftarrow{R} \mathcal{D}_\lambda}[{x''}^* = x'']}{\Pr_{x'' \xleftarrow{R} \mathcal{D}_\lambda}[\mathsf{ProbIndex}({x''}^*, 3\kappa) = \mathsf{ProbIndex}(x'', 3\kappa)]} \cdot \Pr_{x_0 \xleftarrow{R} \mathcal{D}_\lambda}[{x_0}^* = x_0] \cdot \frac{1}{\mathsf{ProbDens}(1^\lambda, x_0)}$$

by simply renaming x_0 to x''

$$= \Pr_{x'' \xleftarrow{R} \mathcal{D}_\lambda}[{x''}^* = x'' | \mathsf{ProbIndex}({x''}^*, 3\kappa) = \mathsf{ProbIndex}(x'', 3\kappa)] \cdot \Pr_{x_0 \xleftarrow{R} \mathcal{D}_\lambda}[{x_0}^* = x_0] \cdot \frac{1}{\mathsf{ProbDens}(1^\lambda, x_0)}$$

again by definition of conditional probability. We can see the final line is the probability density of ${x_0}^*, {x''}^*, p^*$ in Game 8. □

Claim 16. *The distributions output by Game 8 and Game 9 have statistical distance* $\frac{1}{3\kappa} + \mathsf{negl}(\lambda)$.

Proof. Game 9 contains 2 changes. One is purely notational, where x'' is eliminated and we directly test if $\mathsf{ProbIndex}(x_0, 3\kappa) = \mathsf{ProbIndex}(x', 3\kappa)$. Since x'' was drawn from $\mathcal{D}_\lambda$ and x' is drawn from $\mathsf{Sample}(1^\lambda)$, the distribution of x' is statistically close from this change. The other change made here is that we restrict the loop to $9\ln(\mathsf{PDF}_{\max}) \cdot \kappa^2 \cdot \lambda$ iterations rather than an unbounded number. Note that the loop terminating is entirely dependent on sampling x' such that $\mathsf{ProbIndex}(x_0, 3\kappa) = \mathsf{ProbIndex}(x', 3\kappa)$. So we will show that in Game 8, such an x' is found in the first $9\ln(\mathsf{PDF}_{\max}) \cdot \kappa^2 \cdot \lambda$ iterations with probability $\leq \frac{1}{3\kappa} + \mathsf{negl}(\lambda)$.

To see this, we first want to observe that the total number of possible values of $\mathsf{ProbIndex}$ is bounded by

$$\log_{1+\frac{1}{3\kappa}}(\mathsf{PDF}_{\max}) = \ln(\mathsf{PDF}_{\max}) / \ln(((1 + \frac{1}{3\kappa})^{3\kappa \cdot \frac{1}{3\kappa}}))$$

$$= \ln(\mathsf{PDF}_{\max}) \cdot 3\kappa / \ln((1 + \frac{1}{3\kappa})^{3\kappa}) \approx 3\ln(\mathsf{PDF}_{\max}) \cdot \kappa$$

as the $\mathsf{ProbDens}$ function is nonnegative and integral, so $\mathsf{ProbIndex}$ returns $\perp$ or an integer in the range $[0, \ln(\mathsf{PDF}_{\max}) \cdot \kappa)]$. Let the set $A = \{a_1, a_2, \ldots a_q\}$ denote the range of values $\mathsf{ProbIndex}$ can take.

We partition A into sets A_0 and A_1 such that

$$a \in A_0 \Leftrightarrow \Pr\left[\begin{matrix} x_0 \leftarrow \mathcal{D}_\lambda \\ \mathsf{ProbIndex}(x_0, 3\kappa) = a \end{matrix}\right] \leq \frac{1}{9\ln(\mathsf{PDF}_{\max}) \cdot \kappa^2}$$

and similarly

$$a \in A_1 \Leftrightarrow \Pr\left[\begin{matrix} x_0 \leftarrow \mathsf{Sample}(1^\lambda) \\ \mathsf{ProbIndex}(x_0, 3\kappa) = a \end{matrix}\right] > \frac{1}{9\ln(\mathsf{PDF}_{\max}) \cdot \kappa^2}$$

Since A has $3\ln(\mathsf{PDF}_{\max}) \cdot \kappa$ elements, we can bound

$$\Pr\left[x_0 \leftarrow \mathcal{D}_\lambda \mathsf{ProbIndex}(x_0, 3\kappa) \in A_0\right] \leq \frac{3\ln(\mathsf{PDF}_{\max}) \cdot \kappa}{9\ln(\mathsf{PDF}_{\max}) \cdot \kappa^2} = \frac{1}{3\kappa}$$

If we suppose $\mathsf{ProbIndex}(x_0, 3\kappa) \in A_1$, then we can see the probability that $9\ln(\mathsf{PDF}_{\max}) \cdot \kappa^2 \cdot \lambda$ fail to find an x' is lower bounded by

$$\left(1 - \frac{1}{9\ln(\mathsf{PDF}_{\max}) \cdot \kappa^2}\right)^{9\ln(\mathsf{PDF}_{\max})\cdot\kappa^2\cdot\lambda} \approx e^{-\lambda} \in \mathsf{negl}(\lambda)$$

Since we know $\mathsf{ProbIndex}(x_0, 3\kappa) \in A_1$ with probability at least $1 - \frac{1}{3\kappa}$, we can lower bound the probability that this loop terminates within $9\ln(\mathsf{PDF}_{\max}) \cdot \kappa^2 \cdot \lambda$ iterations as $1 - \frac{1}{3\kappa} - \mathsf{negl}(\lambda)$, which bounds the statistical distance between Games 8 and 9 with $\frac{1}{3\kappa} + \mathsf{negl}(\lambda)$. □

Claim 17. *The distributions output by Game 9 and Game 10 have statistical distance* $\mathsf{negl}(\lambda)$.

Proof. In Game 10, we alter the way which we generate the variables x, p, x_0. From Lemma 3, we know the distributions of x_0, p under

$$(x_0, p) \xleftarrow{R} \mathsf{PDF}(\mathcal{D}_\lambda)$$

and

$$(\_, p) \xleftarrow{R} \mathsf{PDF}(\mathcal{D}_\lambda)$$
$$x_0 \leftarrow \mathsf{SampleUniform}(1^\lambda, p)$$

are statistically close. Using this, we can see that the distribution of x_0, x, p induced by Game 9 as

$$(x_0, p) \xleftarrow{R} \mathsf{PDF}(\mathcal{D}_\lambda)$$
$$x \leftarrow \mathsf{SampleUniform}(1^\lambda, p)$$

is statistically close to

$$(\_, p) \xleftarrow{R} \mathsf{PDF}(\mathcal{D}_\lambda)$$
$$x_0 \leftarrow \mathsf{SampleUniform}(1^\lambda, p)$$
$$x \leftarrow \mathsf{SampleUniform}(1^\lambda, p)$$

However, note in the latter distribution, x_0 and x are generated independently, so this is the same as distribution

$$(\_, p) \xleftarrow{R} \mathsf{PDF}(\mathcal{D}_\lambda)$$
$$x \leftarrow \mathsf{SampleUniform}(1^\lambda, p)$$
$$x_0 \leftarrow \mathsf{SampleUniform}(1^\lambda, p).$$

Applying Lemma 3 again, this is statistically close to

$$(x, p) \xleftarrow{R} \mathsf{PDF}(\mathcal{D}_\lambda)$$
$$x_0 \leftarrow \mathsf{SampleUniform}(1^\lambda, p).$$

Which is the distribution of x_0, x, p in Game 10.

□

Claim 18. *The distributions output by Game 10 and Game 11 have statistical distance* $\mathsf{negl}(\lambda)$.

Proof. By Definition 5,

$$(x, p) \xleftarrow{R} \mathsf{PDF}(\mathcal{D}_\lambda)$$

is equivalent to

$$x \xleftarrow{R} \mathcal{D}_\lambda$$

$$p \xleftarrow{R} [0, \mathsf{ProbDens}(1^\lambda, x)].$$

From Lemma 4, this is statistically close to

$$x \leftarrow \mathsf{Sample}'(1^\lambda)$$

$$p \xleftarrow{R} [0, \mathsf{ProbDens}(1^\lambda, x)].$$

□

Combining Claim 8 through Claim 18, we get the total statistical distance of Game 0 and 11 is $\leq \frac{1}{\kappa} + \mathsf{negl}(\lambda)$

□

Combining Claim 7 and Claim 6, we get the explainability of Sample'.

□

Combining Lemma 4 and Lemma 5, we get that Sample' is an explainable sampler for $\{\mathcal{D}_\lambda\}_\lambda$.

□

5.2 Instantiation on Discrete Gaussians

As an example, we can observe that the above gives us an explainable sampler for discrete Gaussians centered at $c(\lambda)$ with standard deviation $\sigma(\lambda)$. For the Sample algorithm, we can use any of the discrete Gaussian samplers present in literature. We can use $\mathsf{ProbDens}(1^\lambda, x) = e^{-\frac{(x-c)^2}{2\sigma^2}}$, which is efficienty computable, and we can let $\mathsf{SampleUniform}(1^\lambda, p)$ simply sample uniformly from integers on the interval $\left[\left\lceil c - \sigma\sqrt{-2\ln(p)}\right\rceil, \left\lfloor c + \sigma\sqrt{-2\ln(p)}\right\rfloor\right]$. Since the only randomness here is sampling uniform integers on a fixed interval, this is easily explainable.

5.3 Impossibility of Generic Sampling without Heavy Element Samplers

We give some evidence on the tightness of the above result by showing an impossibility of a black box construction of an explainable sampler from only a sampler and probability density function to some distribution. This also highlights the inherent need of non-black-box techniques such as indistinguishability obfuscation used to construct universal samplers in [21].

Theorem 5. *There exists a distribution family $\{\mathcal{D}_\lambda\}_\lambda$ such that there does not exist an efficient explainable sampler for $\{\mathcal{D}_\lambda\}_\lambda$ given oracle access to a sampler* Sample *and probability density function* $\mathsf{ProbDens}$.

Proof. To show this, we will give oracles S and P which are a sampler and probability density function for some distribution $\mathcal{D}$ for which no adversary $\mathcal{A}^{S,P}$ can produce a distribution statistically close to $\mathcal{D}$

For each λ, let $S(1^\lambda, \cdot)$ be a random oracle from $\{0,1\}^\lambda \to \{0,1\}^{3\lambda}$. Correspondingly, let $P(1^\lambda, \cdot)$ return 1 for every element in $\{0,1\}^{3\lambda}$ in the range of $S(1^\lambda, \cdot)$ and 0 otherwise.

Lemma 6. *S and P are a sampler and probability density function for the same distribution.*

Proof. We note that we can lower bound the probability that S is injective with $\left(1 - \frac{2^\lambda}{2^{3\lambda}}\right)^{2^{3\lambda}} \approx \left(\frac{1}{e}\right)^{2^\lambda} \in \mathsf{negl}(\lambda)$. Now if this is true, then S is uniform on its range, which is exactly the distribution P describes. □

Lemma 7. *S is a one way function - i.e. for all PPT adversaries $\mathcal{A}^{S,P}$*

$$\Pr\left[\begin{array}{c} r \xleftarrow{R} \{0,1\}^\lambda \\ x \leftarrow S(1^\lambda, r) \\ r' \leftarrow \mathcal{A}^{S,P}(1^\lambda, x) \\ S(1^\lambda, r') = x \end{array}\right] \leq \mathsf{negl}(\lambda)$$

Proof. We will proceed with a short sequence of games argument

---

Game 0

- The challenger $\mathcal{C}$ generates $r \xleftarrow{R} \{0,1\}^\lambda$ and gives adversary $x = S(1^\lambda, r)$.
- The adversary $\mathcal{A}$ submits queries s_i, p_i to S and P which the challenger $\mathcal{C}$ responds with $S(1^\lambda, s_i)$ or $P(1^\lambda, p_i)$ respectively.
- $\mathcal{A}$ submits r'.
- $\mathcal{A}$ wins if $x = S(1^\lambda, r')$.

Game 1

- The challenger generates a random $r \xleftarrow{R} \{0,1\}^\lambda$, $x \xleftarrow{R} \{0,1\}^{3\lambda}$ and reprograms $P(x) = 1$ and $S(1^\lambda, r) = x$.
- The adversary $\mathcal{A}$ submits queries s_i, p_i to S and P which the challenger $\mathcal{C}$ responds with $S(1^\lambda, s_i)$ or $P(1^\lambda, p_i)$ respectively.
- $\mathcal{A}$ submits r'.
- $\mathcal{A}$ wins if $x = S(1^\lambda, r')$.

Game 2

- The challenger generates a random ~~$r \xleftarrow{R} \{0,1\}^\lambda$,~~ $x \xleftarrow{R} \{0,1\}^{3\lambda}$ and reprograms $P(x) = 1$, ~~$S(1^\lambda, r) = x$~~.
- The adversary $\mathcal{A}$ submits queries s_i, p_i to S and P which the challenger $\mathcal{C}$ responds with $S(1^\lambda, s_i)$ or $P(1^\lambda, p_i)$ respectively.
- $\mathcal{A}$ submits r'.
- $\mathcal{A}$ wins if $x = S(1^\lambda, r')$.

Claim 19. $|\Pr[\mathcal{A}\ \textit{wins Game 0}] - \Pr[\mathcal{A}\ \textit{wins Game 1}]| = \mathsf{negl}(\lambda)$

Proof. Since S has uniform and independent entries, reprogramming a single entry with a uniformly random value does not change its distribution. P only differs on the image of $S(1^\lambda, r)$ before programming. However, since this is uniformly random and now independent of S, with only polynomially many queries, the probability $\mathcal{A}$ queries it is $\frac{\mathsf{poly}(\lambda)}{2^{3\lambda}} = \mathsf{negl}(\lambda)$ □

Claim 20. $|\Pr[\mathcal{A}\ \textit{wins Game 1}] - \Pr[\mathcal{A}\ \textit{wins Game 2}]| = \mathsf{negl}(\lambda)$

Proof. These games only differ when $\mathcal{A}$ queries S on r. Since r is uniformly random and chosen independent of x, the probability that r is queried by $\mathcal{A}$ is bound by $\frac{2^\lambda}{2^{3\lambda}}$. □

Claim 21. $\Pr[\mathcal{A}$ *wins Game 2*$] = \mathsf{negl}(\lambda)$

Proof. Since x is no longer reprogrammed into S, the probability that a randomly chosen element is in the image of S is at most $\frac{2^\lambda}{2^{3\lambda}} = \mathsf{negl}(\lambda)$. □

□

Lemma 8. *Let $\mathcal{A}^{S,P}(1^\lambda)$ be a randomized PPT algorithm that outputs $x \in range(S(1^\lambda, \cdot))$ with probability p. Then $\mathcal{A}$ queries S on an r such that $S(1^\lambda, r) = x$ with probability at least $p - \mathsf{negl}(\lambda)$.*

Proof. We will proceed with a short sequence of games argument

---

Game 0

- The adversary $\mathcal{A}$ submits queries s_i, p_i to S and P which the challenger $\mathcal{C}$ responds with $S(1^\lambda, s_i)$ or $P(1^\lambda, p_i)$ respectively.
- $\mathcal{A}$ submits x.
- $\mathcal{A}$ wins if $x \in \text{range}(S(1^\lambda, \cdot))$.

Game 1

- $\mathcal{C}$ implements S and P by sampling $x_i \xleftarrow{\$} 0,1\}^{3\lambda}$ for $i \in 2^\lambda$ and setting $S(1^\lambda, i) = x_i$, and $P(x) = 1 \Leftrightarrow x \in X = \{x_i\}_{i \in 2^\lambda}$
- $\mathcal{A}$ submits queries s_i, p_i to S and P which $\mathcal{C}$ responds with $S(s_i)$ or $P(p_i)$ respectively.
- $\mathcal{A}$ submits x.
- $\mathcal{A}$ wins if $x \in \text{range}(S(1^\lambda, \cdot))$.

Game 2

- The challenger implements S and P by initalizing an empty list $X = \{x_i = \bot\}$.
- The adversary submits queries s_i, p_i to S and P which the challenger responds with $S(1^\lambda, s_i)$ or $P(1^\lambda, p_i)$ respectively.
  - If this is an S query, then $\mathcal{C}$ returns x_{s_i}. If $x_{s_i} = \bot$, then $\mathcal{C}$ samples and sets $x_{s_i} \xleftarrow{R} \{0,1\}^{3\lambda}$ first.
  - If this is a P query and $p_i \notin X$, then $\mathcal{C}$ returns 0, otherwise, $\mathcal{C}$ returns 1.
- $\mathcal{A}$ submits x.
- $\mathcal{C}$ Generates $x_i \xleftarrow{R} \{0,1\}^{3\lambda}$ for all $x_i \in X$ not yet initialized.
- $\mathcal{A}$ wins if $x \in X$.

Game 3

- The challenger implements S and P by initalizing an empty list $X = \{x_i = \perp\}$.
- The adversary submits queries s_i, p_i to S and P which the challenger responds with $S(1^\lambda, s_i)$ or $P(1^\lambda, p_i)$ respectively.
  - If this is an S query, then $\mathcal{C}$ returns x_{s_i}. If $x_{s_i} = \perp$, then $\mathcal{C}$ samples and sets $x_{s_i} \xleftarrow{R} \{0,1\}^{3\lambda}$ first.
  - If this is a P query and $p_i \notin X$, then $\mathcal{C}$ returns 0, otherwise, $\mathcal{C}$ returns 1.
- $\mathcal{A}$ submits x.
- ~~$\mathcal{C}$ Generates $x_i \xleftarrow{R} \{0,1\}^{3\lambda}$ for all $x_i \in X$ not yet initialized.~~
- $\mathcal{A}$ wins if $x \in X$.

Claim 22. $|\Pr[\mathcal{A} \text{ wins Game } 0] - \Pr[\mathcal{A} \text{ wins Game } 1]| = 0$

Proof. This game represent only a notational change. Since S is a random function, every element is generated independently randomly □

Claim 23. $|\Pr[\mathcal{A} \text{ wins Game } 1] - \Pr[\mathcal{A} \text{ wins Game } 2]| = \mathsf{negl}(\lambda)$

Proof. Note that the list X is generated in the exact same way, albeit not at the same time. Games 1 and 2 only differ in that p_i is queries on X while some $x_i = \perp$. However, for each p_i, we can upper bound the probability that any of the $\leq 2^\lambda$ randomly generated x_i after said query returns p_i with $\frac{2^\lambda}{2^{3\lambda}} \in \mathsf{negl}(\lambda)$. Union bounding over all polynomially many p_i still gives $\mathsf{negl}(\lambda)$. □

Claim 24. $|\Pr[\mathcal{A} \text{ wins Game } 2] - \Pr[\mathcal{A} \text{ wins Game } 3]| = \mathsf{negl}(\lambda)$

Proof. In this game, X is no longer the full range of S. These games only differ when x is equal to one of the x_i generated in the deleted step. Similar to the previous claim, we can bound this probability $\frac{2^\lambda}{2^{3\lambda}} \in \mathsf{negl}(\lambda)$. □

By Game 3, X only contains images to queries $\mathcal{A}$ made to S, so combining the above claims gives us our lemma statement.

□

We observe that if there exists an $\mathsf{Sample}'^{S,P}(1^\lambda; r)$ algorithm for the distribution defined by S, P, then necessarily, it needs to output $x \in \text{range}(S(1^\lambda, \cdot))$ with all but negligible probability to be statistically close. Suppose for sake of contradiction that $\mathsf{Sample}'^{S,P}$ had an $\mathsf{Explain}'^{S,P}$ algorithm. Then consider the following adversary for the one way function game in Lemma 7:

$\mathcal{A}^{S,P}(1^\lambda, x)$

- Let $r = \mathsf{Explain}'^{S,P}(1^\lambda, 11, x)$
- Run $\mathsf{Sample}'^{S,P}(1^\lambda; r)$ and record oracle queries to S.
- If S is run on some y such that $S(y) = x$, then return y. Otherwise return $\perp$.

Recall from Definition 3 that the statistical distance between the $x = \mathsf{Sample}'^{S,P}(1^\lambda), r = \mathsf{Explain}'^{S,P}(1^\lambda, 11, x)$ and $r \xleftarrow{R} \{0,1\}^\lambda, x = \mathsf{Sample}'(1^\lambda; r')$ is $\frac{1}{2}$. Recall in the one way function game, x is sampled from S, so by the definition of a sampler, the x used in $\mathcal{A}$ has statistical distance at most $\frac{1}{2} + \mathsf{negl}(\lambda)$ from $x = S(1^\lambda, r) \in \text{range}(S(1^\lambda, \cdot))$. Thus by Lemma 8, such a y exists with probability at least $\frac{1}{2} - \mathsf{negl}(\lambda)$. However, this means $\mathcal{A}$ wins the game in Lemma 7 with noticeable probability, a contradiction. We can conclude that such an $\mathsf{Explain}'$ algorithm does not exist.

□

6 Explaining Discrete Gaussian Samplers

In this section we show an explanation algorithm for the Miccancio-Walter [29] Gaussian Sampling algorithm. This algorithm follows a series of works improving the practicality of discrete Gaussian sampling [10, 12, 15, 30, 25, 2, 24, 33, 4, 22, 32]. This usually comes in the form of some combination of decreased runtime (in either an offline or online phase), decreased memory usage, and decreased entropy usage. While we could simply apply our transformation of Section 5 to the MW sampler to get explainability, this may not in general preserve special properties of many samplers that could be leveraged, and would add an additional overhead for computing the probability density function. In this section we will show a tailored approach to make it explainable as is.

We begin by giving an adapted definition for explainable discrete Gaussian samplers that explicitly allows for the standard deviation and center of the distribution to be given as parameters. The syntax from the previous sections restricts these to be functions of the security parameter λ. We then describe the explainability algorithm for the Miccancio-Walter discrete Gaussian sampler.

Discrete Gaussians

To talk specifically about discrete Gaussians, we will modify need to expand the definition of distribution samplers to accomodate parameterization. The theorems and their proofs shown in the previous sections translate analogously to the definitions here. For completeness, the proofs will be available in the appendices.

Definition 6. *We say an algorithm* $\mathsf{SampleDG}(1^\lambda, c, \sigma; r)$ *is an* $f_\sigma(\lambda)$*-discrete Gaussian sampler for a discrete Gaussians if for all* $c \in [0,1]$ *and* $\sigma \leq f_\sigma(\lambda)$*,* Sample *runs in* $\mathsf{poly}(\lambda)$ *time and the output of* $\mathsf{SampleDG}(1^\lambda, c, \sigma; r)$ *is statistically close to the discrete Gaussian on* $\mathbb{Z} + c$ *centered at* 0 *with standard deviation* σ*. Note that this is equivalent to producing discrete Gaussians samples on* $\mathbb{Z}$ *centered at any* $c' \in \exp(\lambda)$*, as we can simply generate a sample on* $\mathbb{Z} + (-c \mod 1)$ *centered at* 0 *and add* c*.*

Definition 7. *Analogously, we say a* $\mathsf{SampleDG}$ *algorithm is explainable if there exists a (possibly randomized) algorithm* $\mathsf{ExplainDG}(1^\lambda, 1^\kappa, x, c, \sigma)$ *such that* $\mathsf{ExplainDG}$ *runs in* $\mathsf{poly}(\lambda, \kappa)$ *time, and there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that for all* $c \in [0,1]$*,* $\sigma \leq f_\sigma(\lambda)$*, the statistical distance between the following 2 distributions is at most* $\frac{1}{\kappa} + \mathsf{negl}(\lambda)$*.*

Game A	Game B
• $r \xleftarrow{R} \{0,1\}^n$.	• $x \leftarrow \mathsf{SampleDG}(1^\lambda, c, \sigma)$.
• $x \leftarrow \mathsf{SampleDG}(1^\lambda, c, \sigma; r)$.	• $r \leftarrow \mathsf{ExplainDG}(1^\lambda, 1^\kappa, x, c, \sigma)$.
• Return r, x.	• Return r, x.

While Theorem 4 does imply the existence of an explainable discrete Gaussian sampler, oftentimes, it may be of interest whether existing implementations using particular sampling algorithms are explainable. We'll use this MW17 as an example of a practical discrete Gaussian sampler and prove its explainability, along the way hopefully demonstrating some techniques useful for proving explainability of other sampling algorithms.

6.1 Miccancio-Walter '17

Here, we look at a fairly recent method of sampling discrete Gaussians presented in [29]. Informally, this is done by taking a sampler $\mathsf{SampleBase}$ to discrete Gaussians with small standard deviation, for which there are easier and more efficient to compute, then taking particular linear combinations to create discrete Gaussian samples with large standard deviation in $\mathsf{SampleI}$, before employing a randomized rounding technique in $\mathsf{SampleC}$ to sample from a specific coset of $\mathbb{Z}$.

In comparison to previous works, this has the advantage of better performance, as well as taking time independent of the sample value. In addition, much of the computation can be done offline, without knowing the standard deviation or center an algorithm which calls the sampler may request.

$\mathsf{SampleI}(1^\lambda, i)$

- If $i = 0$
  - $x \leftarrow \mathsf{SampleBase}(1^\lambda, 0, s_0)$
  - Return x
- $x_1 \leftarrow \mathsf{SampleI}(1^\lambda, i-1)$
- $x_2 \leftarrow \mathsf{SampleI}(1^\lambda, i-1)$
- $y = z_i x_1 + \max(1, z_i - 1)x_2$
- Return y

$\mathsf{SampleC}(1^\lambda, c, k)$

- If $k = 0$
  - Return 0.
- $g \leftarrow 2^{-k+1} \cdot \mathsf{SampleBase}(1^\lambda, 2^{k-1}c, s_0)$
- Return $g + \mathsf{SampleC}(1^\lambda, c - g, k - 1)$

$\mathsf{SampleDG}(1^\lambda, c, s)$[5]

- $x \leftarrow \mathsf{SampleI}(m)$
- $K \leftarrow \sqrt{s^2 - \bar{s}_k^2}/s_m$
- $c' \leftarrow \lfloor c + Kx \rceil_k$
- $y \leftarrow \mathsf{SampleC}(1^\lambda, c', k)$
- Return y

We remark that the notation of the above algorithms has been slightly altered to better fit with our definition of a discrete Gaussian sampler. To align with our definition of statistical closeness, we consider $k \in \Theta(\lambda)$. As in the original construction, we let s_i denote the exact standard deviation of $\mathsf{SampleI}$, z_i denote the multipliers used in $\mathsf{SampleI}$, and $\bar{s}_k$ to be the standard deviation 'added' by $\mathsf{SampleC}$, whose explicit values are determined recursively by the equations below. The exact formula is given as a function of the smoothing parameter $\eta_\epsilon(\mathbb{Z})$ of the discrete Gaussian, which, for our purposes, we will only bound as a value $\in O(\log(\lambda))$. Similarly, $m = m(\lambda)$ is a parameter controlling the precision controlling the maximum standard deviation $\mathsf{SampleDG}$ can generate, which we can think of as a value bounded by $\log(\lambda) + O(1)$. Readers of [29] may notice the lack of a base parameter b which controls the 'base' for (e.g. binary, decimal) which randomized rounding occurs. This has been fixed to 2 for simplicity.

$$s_0 = \sqrt{2}\eta_\epsilon(\mathbb{Z})$$

$$z_i = \left\lfloor \frac{s_{i-1}}{\sqrt{2}\eta_\epsilon(\mathbb{Z})} \right\rfloor$$

$$s_i^2 = \left(z_i^2 + \max((z_i - 1)^2, 1)\right) s_{i-1}^2$$

$$\bar{s}_k = s_0 \cdot \left(\sqrt{\sum_{i=0}^{k-1} 2^{-2i}} \right)$$

[5] This was referred to as $\mathsf{SampleZ}$ in [29]

Theorem 6. *Suppose* $\mathsf{SampleBase}(1^\lambda, c)$ *is an* $O(\log(\lambda))$*-discrete Gaussian sampler. Then* $\mathsf{SampleDG}(1^\lambda, c, s; r)$ *is an explainable* $\exp(\lambda)$*-discrete Gaussian sampler for the family of discrete Gaussians centered at* c *with standard deviation* s*.*

Proof.

Lemma 9. *Suppose* $\mathsf{SampleBase}(1^\lambda, c)$ *is an* $O(\log(\lambda))$*-discrete Gaussian sampler. Then for* $i \leq \log(\lambda) + O(1)$*,* $\mathsf{SampleI}(1^\lambda, i)$ *is a sampler for the family of discrete Gaussians centered at* 0 *with standard deviation* $\omega(2^{2^i})$*.*

See [29] for proof.

Lemma 10. *Suppose* $\mathsf{SampleBase}(1^\lambda, c)$ *is an* $O(\log(\lambda))$*-discrete Gaussian sampler. Then* $y = \mathsf{SampleC}(1^\lambda, c, k)$ *is a sampler for the distribution of discrete Gaussians on* $c + \mathbb{Z}$ *with standard deviation* $\bar{s}_k$*.*

See [29] for proof.

Corollary 7. *Suppose* $\mathsf{SampleBase}(1^\lambda, c)$ *is an* $O(\log(\lambda))$*-discrete Gaussian sampler. Let* c' *be a sample from a discrete Gaussians on* $\mathbb{Z}/2^k$ *with standard deviation* $\sqrt{\sigma^2 - \bar{s}_k^2}$ *and center* c*. Then for* $k \in \Theta(\lambda)$*, in* $\mathsf{SampleC}(1^\lambda, c', k)$*, the output to the recursive call to* $\mathsf{SampleC}(1^\lambda, c'', i)$ *is statistically close to the discrete Gaussian on* $\mathbb{Z} + c''$ *with standard deviation* $\sqrt{\sigma^2 - \bar{s}_i^2}$*.*

Proof. This follows from a straightforward induction argument on i. □

Lemma 11. *Suppose* $\mathsf{SampleBase}(1^\lambda, c)$ *is an* $O(\log(\lambda))$*-discrete Gaussian sampler. Then* $\mathsf{SampleDG}(1^\lambda, c, \sigma)$ *is a* $\exp(\lambda)$*-discrete Gaussian sampler.*

See [29] for proof.

Lemma 12. *Any* $O(\log(\lambda))$*-discrete Gaussian sampler is explainable.*

Proof. Note that we can bound the probability a sample is outside $\log(\lambda)$ standard deviations with $2 \cdot \sum_{i=\log(\lambda)\sigma}^{\infty} e^{-\frac{i^2}{2\sigma}}$ Since this decays exponentially, this is $\leq O(e^{-\log(\lambda)^2}) \in O(\lambda^{-\log(\lambda)})$. Thus, we can take the set S_λ to be the set of elements within $\log(\lambda)$ standard deviations of the center. By Corollary 3 from Section 4, this is explainable. □

Lemma 13. *Suppose* $\mathsf{SampleBase}(1^\lambda, c)$ *is an* $O(\log(\lambda))$*-discrete Gaussian sampler. Then* $\mathsf{SampleI}(1^\lambda, i)$ *is an* ***explainable*** *sampler for the family of discrete Gaussians centered at* 0 *with standard deviation* $\omega(2^{2^i})$*.*

Proof. We can prove this inductively on i. For $i = 0$, $\mathsf{SampleI}$ is simply a call to $\mathsf{SampleBase}$, by Lemma 12 is explainable, so has associated $\mathsf{ExplainBase}$ algorithm. Suppose $\mathsf{SampleI}(1^\lambda, i)$ samples from a 0 centered discrete Gaussian with standard deviation σ_i and has a corresponding $\mathsf{Explain}_i(1^\lambda, 1^\kappa, x)$ algorithm. In addition, suppose we have access to a (not necessarily explainable) $\exp(\lambda)-$discrete Gaussian sampler $\mathsf{SampleDG}(1^\lambda, c, \sigma)$. Then consider the following $\mathsf{Explain}_{i+1}$ algorithm:

$\mathsf{Explain}_0(1^\lambda, 1^\kappa, x) = \mathsf{ExplainBase}(1^\lambda, 1^\kappa, x)$

$\mathsf{Explain}_{i+1}(1^\lambda, 1^\kappa, x)$

- $t \leftarrow \mathsf{SampleDG}(1^\lambda, -\frac{x - 2xz_i}{2z_i^2 - 2z_i + 1}, \frac{\sigma_i}{2z_i^2 - 2z_i + 1}) + \frac{x - 2xz_i}{2z_i^2 - 2z_i + 1}$
- $x_1 = x + (z_i - 1) \cdot t$
- $x_2 = -x - z_i \cdot t$
- $r_1 \leftarrow \mathsf{Explain}_i(1^\lambda, 1^{2\kappa}, x_1)$
- $r_2 \leftarrow \mathsf{Explain}_i(1^\lambda, 1^{2\kappa}, x_2)$
- Return (r_1, r_2)

First, note that the only randomness used in $\mathsf{SampleI}(1^\lambda, i+1)$ is the randomness is the randomness used by the recursive calls to $\mathsf{SampleI}(1^\lambda, i)$, so it suffices to generate randomness to explain these calls. Suppose we fix some sum x to explain. We can first that since $z_i, z_i - 1$ are relatively prime, the set of possible x_1 and x_2 such that $z_i x_1 + (z_i - 1)x_2 = x$ can be parameterized by

$$x_1 = x + (z_i - 1) \cdot t$$

$$x_2 = -x - z_i \cdot t$$

for $t \in \mathbb{Z}$.

Recall the exact probability of a particular x_1, x_2 being chosen is is

$$exp\left(-\frac{x_1^2}{2\sigma_i^2}\right) \cdot exp\left(-\frac{x_2^2}{2\sigma_i^2}\right)$$

$$= exp\left(-\frac{x_1^2 + x_2^2}{2\sigma_i^2}\right)$$

$$= exp\left(-\frac{(x + (z_i - 1)t)^2 + (x + z_i \cdot t)^2}{2\sigma_i^2}\right)$$

$$= exp\left(-\frac{\left((2z_i^2 - 2z_i + 1)t - (x - 2xz_i)\right)^2 + \frac{x^2}{2z_i(z_i-1)+1}}{2\sigma_i^2}\right)$$

$$= exp\left(-\frac{\left((2z_i^2 - 2z_i + 1)t - (x - 2xz_i)\right)^2}{2\sigma_i^2}\right) \cdot \exp\left(\frac{\frac{x^2}{2z_i(z_i-1)+1}}{2\sigma_i^2}\right)$$

Note that since the second term is a constant multiplicative factor with respect to t, we can ignore it when considering the conditional probability on a fixed x and z_i. But we can observe the first term with respect to t is simply a discrete Gaussian centered at $\frac{x-2xz_i}{2z_i^2-2z_i+1}$ with standard deviation $\frac{\sigma_i}{2z_i^2-2z_i+1}$. So to sample on the joint distribution of x_1, x_2 conditional on fixed sum x, it suffices to sample from the one-dimensional discrete Gaussian variable t. This is a discrete analogue to the well known fact that a projection of a multivariate Gaussian onto any subspace is still Gaussian. Using this fact, in addition to Lemma 9, this tells us that the distributions:

Distribution 1

- $x_1 \leftarrow \mathsf{SampleI}(1^\lambda, i)$.
- $x_2 \leftarrow \mathsf{SampleI}(1^\lambda, i)$.
- Return x_1, x_2.

Distribution 2

- $x \leftarrow \mathsf{SampleI}(1^\lambda, i+1)$.
- $t \leftarrow \mathsf{SampleDG}(1^\lambda, \frac{x-2xz_i}{2z_i^2-2z_i+1}, \frac{\sigma_i}{2z_i^2-2z_i+1})$.
- $x_1 = x + (z_i - 1) \cdot t$.
- $x_2 = -x - z_i \cdot t$.
- Return x_1, x_2.

are statistically close. Using the definition of $\mathsf{Explain}_i$, we know that running $r_i \leftarrow \mathsf{Explain}_i(1^\lambda, 1^{2\kappa}, x_i)$ on the output of Distribution 1 is has statistical distance $\frac{1}{2\kappa}$ from uniform random for each of x_1, x_2, so we can union bound the total statistical distance with $\frac{1}{\kappa} + \mathsf{negl}(\lambda)$. On the other hand, we can observe that running $\mathsf{Explain}_i$ on the output of Distribution 2 corresponds to an execution of $\mathsf{Explain}_{i+1}(1^\lambda, 1^\kappa, x)$ in the Definition 3 Game B. Since these games are statistically close, we can bound the total statistical distance between $\mathsf{Explain}_{i+1}(1^\lambda, 1^\kappa, x)$ and uniform random with $\frac{1}{\kappa} + \mathsf{negl}(\lambda)$.

We can also observe that $\mathsf{Explain}_i$ is efficient, since $i < \log(\lambda)$, the number of recursive calls to $\mathsf{Explain}_0$ is bounded by $O(\lambda)$, and the κ value called on in lower order recursive calls is at most $\lambda \cdot \kappa$.

□

Lemma 14. *Suppose* $\mathsf{SampleBase}(1^\lambda, c)$ *is an* $O(\log(\lambda))$*-discrete Gaussian sampler. Let* $\mathsf{SampleDG}_{2^k}(1^\lambda, c, \sigma)$ *be an explainable sampler for discrete Gaussians on* $\mathbb{Z}/2^k$ *with standard deviation* σ *and center* c*. Then for* $k \in O(\lambda)$*,* $\mathsf{SampleC}'(1^\lambda, c, \sigma) = \mathsf{SampleC}(1^\lambda, \mathsf{SampleDG}_{2^k}(1^\lambda, c, \sqrt{\sigma^2 - \bar{s}_k^2}), k)$ *is an explainable discrete Gaussian sampler.*

Proof. From Corollary 7, we can characterize the output of the recursive call to $\mathsf{SampleC}$ as a discrete Gaussian. Also, recall that from Lemma 12, $\mathsf{SampleBase}$ is explainable, with an associated $\mathsf{ExplainBase}$. Similar to the previous lemma, we can prove construct an 'explanation' for $\mathsf{SampleC}'$ recursively.

$\mathsf{CosetProb}(1^\lambda, \sigma, h)$

- Sample $z \leftarrow \mathsf{SampleC}'(1^\lambda, h, \sqrt{\sigma^2 + \bar{s}_{k-1}^2})$
- Sample $z' \in [z, z+1)$
- Return 1 with probability $exp\left(-\frac{z'^2}{2(\sigma^2+\bar{s}_{k-1}^2)}\right) / \left(\lambda^3 \cdot exp\left(\frac{z^2}{2(\sigma^2+\bar{s}_{k-1}^2)}\right)\right)$, otherwise return 0.

$\mathsf{ExplainC}'_k(1^\lambda, x, c, \sigma)$

- If $k = 0$
  - Set $c_0 = c$
  - Return $\{c_0\}$
- Run λ^7 times:
  - Sample $g_k \leftarrow \mathsf{SampleDG}_{2^{k-1}}\left(1^\lambda, -\frac{x(2^{-k+1}s_0)^2}{(\sigma^2+2s_{k-1}^2+(2^{-k+1}s_0)^2)}, \sqrt{\frac{(2^{-k+1}s_0)^2(\sigma^2+2s_{k-1}^2)}{\sigma^2+2s_{k-1}^2+(2^{-k+1}s_0)^2}}\right) + \frac{x(2^{-k+1}s_0)^2}{(\sigma^2+2s_{k-1}^2+(2^{-k+1}s_0)^2)}$.
  - If $\mathsf{CosetProb}(1^\lambda, \sigma, c - g) = 1$:
    - $\{r, g_1 \dots g_{k-1}\} \leftarrow \mathsf{ExplainC}'_{k-1}(1^\lambda, c - g_k, \sqrt{\sigma^2 + \bar{s}_{k-1}^2})$.
    - Return $\{c_0, g_1 \dots g_{k-1}, g_k\}$.
- Return $\perp$.

$\mathsf{ExplainC}'(1^\lambda, 1^\kappa, x, c, \sigma)$

- $\{c_0, g_1, \dots g_k\} \leftarrow \mathsf{ExplainC}'_k(1^\lambda, x, c, \sigma)$
- Set $r_0 = \mathsf{ExplainDG}_{2^k}(1^\lambda, 1^{(k+1)\kappa}, c_0, c, \sigma)$
- For $i \in [k]_\mathbb{Z}$
  - Set $r_i = \mathsf{ExplainBase}(1^\lambda, 1^{(k+1)\kappa}, 2^{i-1}g_i, 2^{i-1}(c_0 - \sum_{j>i} g_j), 2^{-i} \cdot s_0)$
- Return $\{r_0, \dots r_k\}$

Claim 25. $\mathsf{CosetProb}(1^\lambda, \sigma, h)$ *returns* 1 *with probability*

$$\frac{\left(\int_{-\infty}^{\infty} exp\left(-\frac{i^2}{2(\sigma^2+\bar{s}_{k-1}^2)}\right)\right)}{\lambda^3 \cdot \left(\sum_{j \in \mathbb{Z}+h} exp\left(-\frac{j^2}{2(\sigma^2+\bar{s}_{k-1}^2)}\right)\right)}$$

Proof. From Corollary 7, $z \leftarrow \mathsf{SampleC}'(1^\lambda, h, \sqrt{\sigma^2 + \bar{s}_{k-1}^2})$ will return a discrete Gaussian on $\mathbb{Z}+h$ with standard deviation $\sqrt{\sigma^2 - \bar{s}_{k-1}^2}$, which, by definition, has probability density proportional to $exp\left(\frac{z^2}{2\left(\sigma^2+\bar{s}_{k-1}^2\right)}\right)$.

We can compute the total probability of $\mathsf{CosetProb}$ returning 1 by conditioning on the value z takes

$$\Pr[\mathsf{CosetProb}(1^\lambda, \sigma, h) = 1] = \sum_{i \in \mathbb{Z}+h} \Pr[\mathsf{CosetProb}(1^\lambda, \sigma, h) = 1 | z = i] \cdot \Pr[z = i]$$

$$= \sum_{i \in \mathbb{Z}+h} \int_i^{i+1} \frac{exp\left(-\frac{j^2}{2\left(\sigma^2+\bar{s}_{k-1}^2\right)}\right)}{\left(\lambda^3 \cdot exp\left(\frac{z^2}{2\left(\sigma^2+\bar{s}_{k-1}^2\right)}\right)\right)} dj \cdot \frac{exp\left(\frac{i^2}{2\left(\sigma^2+\bar{s}_{k-1}^2\right)}\right)}{\sum_{i \in \mathbb{Z}+h} exp\left(\frac{i^2}{2\left(\sigma^2+\bar{s}_{k-1}^2\right)}\right)}$$

$$= \sum_{i \in \mathbb{Z}+h} \int_i^{i+1} exp\left(-\frac{j^2}{2\left(\sigma^2 + \bar{s}_{k-1}^2\right)}\right) dj \cdot \frac{1}{\left(\lambda^3 \cdot exp\left(\frac{z^2}{2\left(\sigma^2+\bar{s}_{k-1}^2\right)}\right)\right)} \cdot \frac{\sum_{i \in \mathbb{Z}+h} exp\left(\frac{i^2}{2\left(\sigma^2+\bar{s}_{k-1}^2\right)}\right)}{\sum_{i \in \mathbb{Z}+h} exp\left(\frac{i^2}{2\left(\sigma^2+\bar{s}_{k-1}^2\right)}\right)}$$

$$= \int_{-\infty}^{\infty} exp\left(-\frac{j^2}{2\left(\sigma^2 + \bar{s}_{k-1}^2\right)}\right) dj \cdot \frac{1}{\left(\lambda^3 \cdot exp\left(\frac{z^2}{2\left(\sigma^2+\bar{s}_{k-1}^2\right)}\right)\right)}$$

□

Claim 26. $\mathsf{CosetProb}(1^\lambda, \sigma, h)$ *returns* 1 *with probability* $\geq \lambda^{-6}$.

Proof. From Corollary 7, $\mathsf{SampleC}'(1^\lambda, h, \sqrt{\sigma^2 + \bar{s}_{k-1}^2})$ will return a discrete Gaussian on $\mathbb{Z}+h$ with standard deviation $\sqrt{\sigma^2 - \bar{s}_{k-1}^2}$, which, by definition, has probability density proportional to $exp\left(\frac{z^2}{2\left(\sigma^2+\bar{s}_{k-1}^2\right)}\right)$

Since this rounding changes varies z by at most 1, we can bound

$$exp\left(-\frac{z^2 + 2|z| + 1}{2\left(\sigma^2 + \bar{s}_{k-1}^2\right)}\right) \leq exp\left(-\frac{z'^2}{2\left(\sigma^2 + \bar{s}_{k-1}^2\right)}\right) \leq exp\left(-\frac{z^2 - 2|z| + 1}{2\left(\sigma^2 + \bar{s}_{k-1}^2\right)}\right)$$

Since we know that with all but negligible probability, $|z| \leq \ln(\lambda) \cdot \sqrt{\sigma^2 + \bar{s}_{k-1}^2}$, this bound can be simplified to

$$e^{-3\ln(\lambda)} \cdot exp\left(-\frac{z^2}{2\left(\sigma^2 + \bar{s}_{k-1}^2\right)}\right) \leq exp\left(-\frac{z'^2}{2\left(\sigma^2 + \bar{s}_{k-1}^2\right)}\right) \leq e^{3\ln(\lambda)} \cdot exp\left(-\frac{z^2 - 2|z| + 1}{2\left(\sigma^2 + \bar{s}_{k-1}^2\right)}\right)$$

so we can bound the probability computed $\in [\frac{1}{\lambda^6}, 1]$. □

Much like the explanation for $\mathsf{SampleI}$, for each level, since we know the output behavior of $\mathsf{SampleC}$ at each level thanks to Lemma 10, we can recursively construct an explanation for each level using the joint distribution for said level. The probability of a particular pair $g, x - g$ being sampled by $\mathsf{SampleC}$ can be computed by definition of discrete Gaussian to be

$$exp\left(-\frac{g^2}{2\left(2^{-k+1}s_0\right)^2}\right) \cdot exp\left(-\frac{(x-g)^2}{2\left(\sigma^2 + \bar{s}_{k-1}^2\right)}\right) /$$

$$\left(\sum_{i \in \mathbb{Z}/2^{k-1}+c} exp\left(-\frac{(c-i)^2}{2\left(2^{-k+1}s_0\right)^2}\right)\right)\left(\sum_{j \in \mathbb{Z}+c-g} exp\left(-\frac{j^2}{2\left(\sigma^2 + \bar{s}_{k-1}^2\right)}\right)\right)$$

First, we can consider sampling from a distribution with probability density

$$exp\left(-\frac{g^2}{2\left(2^{-k+1}s_0\right)^2}\right)\cdot exp\left(-\frac{(x-g)^2}{2\left(\sigma^2+\bar{s}_{k-1}^2\right)}\right)$$

which we can do very much analogously to $\mathsf{SampleI}$ - by observing that the numerator of this probability is a quadratic in g, so by converting it to vertex form, we can observe that even conditional on a fixed x value, it suffices to sample g from a discrete Gaussian distribution.

$$\begin{aligned}
&exp\left(-\frac{g^2}{2\left(2^{-k+1}s_0\right)^2}\right)\cdot exp\left(-\frac{(x-g)^2}{2\left(\sigma^2+\bar{s}_{k-1}^2\right)}\right)\\
&=exp\left(-\frac{g^2(\sigma^2+2s_{k-1}^2)+(x-g)^2\left(2^{-k+1}s_0\right)^2}{2\left(2^{-k+1}s_0\right)^2(\sigma^2+2s_{k-1}^2)}\right)\\
&=exp\left(-\frac{g^2(\sigma^2+2s_{k-1}^2+(2^{-k+1}s_0)^2)-g(2x(2^{-k+1}s_0)^2)+x^2\left(2^{-k+1}s_0\right)^2}{2\left(2^{-k+1}s_0\right)^2(\sigma^2+2s_{k-1}^2)}\right)
\end{aligned}$$

$$=exp\left(-\frac{\left(g-\frac{x(2^{-k+1}s_0)^2}{(\sigma^2+2s_{k-1}^2+(2^{-k+1}s_0)^2)}\right)^2}{2\cdot\frac{(2^{-k+1}s_0)^2(\sigma^2+2s_{k-1}^2)}{\sigma^2+2s_{k-1}^2+(2^{-k+1}s_0)^2}}\right)\cdot exp\left(\frac{x^2\left(2^{-k+1}s_0\right)^2}{(2^{-k+1}s_0)^2(\sigma^2+2s_{k-1}^2)}-\frac{(2x(2^{-k+1}s_0)^2)^2}{8\cdot(\sigma^2+2s_{k-1}^2)\cdot\left(2^{-k+1}s_0\right)^2(\sigma^2+2s_{k-1}^2)}\right)$$

Which is just a discrete Gaussian centered at $\frac{x(2^{-k+1}s_0)^2}{(\sigma^2+2s_{k-1}^2+(2^{-k+1}s_0)^2)}$ with standard deviation $\sqrt{\frac{(2^{-k+1}s_0)^2(\sigma^2+2s_{k-1}^2)}{\sigma^2+2s_{k-1}^2+(2^{-k+1}s_0)^2}}$.

However, unlike in $\mathsf{SampleI}$, the denominator is not constant due to the dependence of the term

$$\left(\sum_{j\in\mathbb{Z}+c-g}exp\left(-\frac{j^2}{2\left(\sigma^2+\bar{s}_{k-1}^2\right)}\right)\right)$$

on g. However, we correct for this difference using an $\mathsf{CosetProb}$ to apply rejection sampling to correct this probability - after sampling a candidate g solutionl by Claim 25, we accept it with probability

$$\frac{\left(\int_{-\infty}^{\infty}exp\left(-\frac{i^2}{2\left(\sigma^2+\bar{s}_{k-1}^2\right)}\right)\right)di}{\lambda^3\cdot\left(\sum_{j\in\mathbb{Z}+c-g}exp\left(-\frac{j^2}{2\left(\sigma^2+\bar{s}_{k-1}^2\right)}\right)\right)}$$

Since $\mathsf{ExplainC}_k'$ only returns $\perp$ if $\mathsf{CosetProb}$ returns 0 all λ^7 times, we can upper bound the probability $\mathsf{ExplainC}_k'$ returns $\perp$ with $\left(1-\frac{1}{\lambda^6}\right)^{\lambda^7}\approx e^{-\lambda}\in\mathsf{negl}(\lambda)$.

Note that since $\int_{-\infty}^{\infty}exp\left(-\frac{i^2}{2\left(\sigma^2+\bar{s}_{k-1}^2\right)}\right)di$ is independent of g, all the remaining the accepting probability is proportional to

$$\begin{aligned}
&exp\left(-\frac{g^2}{2\left(2^{-k+1}s_0\right)^2}\right)\cdot exp\left(-\frac{(x-g)^2}{2\left(\sigma^2+\bar{s}_{k-1}^2\right)}\right)\\
&=exp\left(-\frac{g^2(\sigma^2+2s_{k-1}^2)+(x-g)^2\left(2^{-k+1}s_0\right)^2}{2\left(2^{-k+1}s_0\right)^2(\sigma^2+2s_{k-1}^2)}\right)\\
&=exp\left(-\frac{g^2(\sigma^2+2s_{k-1}^2+(2^{-k+1}s_0)^2)-g(2x(2^{-k+1}s_0)^2)+x^2\left(2^{-k+1}s_0\right)^2}{2\left(2^{-k+1}s_0\right)^2(\sigma^2+2s_{k-1}^2)}\right)
\end{aligned}$$

, so we can conclude that if $\mathsf{ExplainC}'_k$ does not return $\perp$, it returns the marginal distribution on $c_0, g_1, \dots g_k$ conditioned on x with all but negligible probability - in other words - the following distributions are statistically close -

Distribution 1

- $c_0 \leftarrow \mathsf{SampleDG}_{2^k}(1^\lambda, c, \sqrt{\sigma^2 - \bar{s}_k^2})$.
- For $i \in [k]_{\mathbb{Z}}$.
  - Set $g_i = 2^{-k+1} \cdot \mathsf{SampleBase}(1^\lambda, 2^{k-1}c_{i-1}, s_0)$
  - Set $c_i = c_{i-1} - g_i$
- Return $\{c_0, g_1, \dots g_k\}$.

Distribution 2

- $x \leftarrow \mathsf{SampleC}'(1^\lambda, c, \sigma)$.
- $\{c_0, g_1, \dots g_k\} \leftarrow \mathsf{ExplainC}'_k(1^\lambda, x, c, \sigma)$
- Return $\{c_0, g_1, \dots g_k\}$

By Definition 3, running $\mathsf{ExplainDG}, \mathsf{ExplainBase}$ on the corresponding outputs to Distribution 1 has statistical distance at most $(k+1) \cdot \frac{1}{(k+1)\kappa} + \mathsf{negl}(\lambda)$ from uniformly random, and so should have the same $\frac{1}{\kappa} + \mathsf{negl}(\lambda)$ statistical distance in Distribution 2 as well. However, we can observe this corresponds exactly to using the execution of $\mathsf{ExplainC}'$ in Definition 3 Game B.

We can also see that $\mathsf{ExplainC}'$ is efficient, as since $k \in O(\lambda)$, $\mathsf{ExplainC}'_k$ is bounded by $O(\lambda)$ recursive calls, all with the same parameters. In addition, all calls to $\mathsf{ExplainDG}, \mathsf{ExplainBase}$ incur a $O(\lambda)$ blowup in κ, so are still polynomial. □

From Lemma 13, we know the $\mathsf{SampleI}$ algorithm is an explainable sampler for discrete Gaussians centered at 0 with standard deviation s_m. We know from this that c' is a sample from a discrete Gaussian on $\mathbb{Z}/2^k$ centered at c with standard deviation $\sqrt{s^2 - \bar{s}_k^2}$. Moreover, this is explainable (the only non-reversible step is the rounding to the nearest 2^{-k}, but since $k \in \Theta(\lambda)$, we can pick a uniform preimage to the rounding and still be statistically close). Thus, using Lemma 14, taking $\mathsf{SampleDG}$ to be the first 3 lines of $\mathsf{SampleDG}$, we can conclude $\mathsf{SampleDG}$ is explainable. Combined with Lemma 11, this gives us the statement of Theorem 6. □

References

[1] Shweta Agrawal, Daniel Wichs, and Shota Yamada. Optimal broadcast encryption from LWE and pairings in the standard model. In Rafael Pass and Krzysztof Pietrzak, editors, *Theory of Cryptography - 18th International Conference, TCC 2020, Durham, NC, USA, November 16-19, 2020, Proceedings, Part I*, volume 12550 of *Lecture Notes in Computer Science*, pages 149–178. Springer, 2020.

[2] Carlos Aguilar-Melchor, Martin R. Albrecht, and Thomas Ricosset. Sampling from arbitrary centered discrete gaussians for lattice-based cryptography. In Dieter Gollmann, Atsuko Miyaji, and Hiroaki Kikuchi, editors, *Applied Cryptography and Network Security*, pages 3–19, Cham, 2017. Springer International Publishing.

[3] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, volume 7237 of *Lecture Notes in Computer Science*, pages 483–501. Springer, 2012.

[4] Gilles Barthe, Sonia Belaïd, Thomas Espitau, Pierre-Alain Fouque, Mélissa Rossi, and Mehdi Tibouchi. Galactics: Gaussian sampling for lattice-based constant- time implementation of cryptographic signatures, revisited. pages 2147–2164, 11 2019.

[5] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5, 1993*, pages 62–73. ACM, 1993.

[6] Mihir Bellare and Phillip Rogaway. The exact security of digital signatures - how to sign with RSA and rabin. In Ueli M. Maurer, editor, *Advances in Cryptology - EUROCRYPT '96, International Conference on the Theory and Application of Cryptographic Techniques, Saragossa, Spain, May 12-16, 1996, Proceeding*, volume 1070 of *Lecture Notes in Computer Science*, pages 399–416. Springer, 1996.

[7] Dan Boneh and Matt Franklin. Identity-based encryption from the weil pairing. In Joe Kilian, editor, *Advances in Cryptology — CRYPTO 2001*, pages 213–229, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[8] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In Colin Boyd, editor, *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings*, volume 2248 of *Lecture Notes in Computer Science*, pages 514–532. Springer, 2001.

[9] Zvika Brakerski, David Cash, Rotem Tsabary, and Hoeteck Wee. Targeted homomorphic attribute-based encryption. In Martin Hirt and Adam D. Smith, editors, *Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part II*, volume 9986 of *Lecture Notes in Computer Science*, pages 330–360, 2016.

[10] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '13, page 575–584, New York, NY, USA, 2013. Association for Computing Machinery.

[11] Eric Brier, Jean-Sébastien Coron, Thomas Icart, David Madore, Hugues Randriam, and Mehdi Tibouchi. Efficient indifferentiable hashing into ordinary elliptic curves. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, pages 237–254, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[12] Johannes Buchmann, Daniel Cabarcas, Florian Göpfert, Andreas Hülsing, and Patrick Weiden. Discrete ziggurat: A time-memory trade-off for sampling from a gaussian distribution over the integers. In Tanja Lange, Kristin Lauter, and Petr Lisoněk, editors, *Selected Areas in Cryptography – SAC 2013*, pages 402–417, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[13] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited (preliminary version). In Jeffrey Scott Vitter, editor, *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 209–218. ACM, 1998.

[14] Pratish Datta, Ilan Komargodski, and Brent Waters. Decentralized multi-authority abe for dnfs from lwe. In Kenneth G. Paterson, editor, *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zoagreb, Croatia, October 17-21, 2021. Proceedings*, Lecture Notes in Computer Science. Springer, 2021.

[15] Nagarjun C. Dwarakanath and Steven D. Galbraith. Sampling from discrete gaussians for lattice-based cryptography on a constrained device. *Appl. Algebra Eng., Commun. Comput.*, 25(3):159–180, June 2014.

[16] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1986.

[17] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 537–554. Springer, 1999.

[18] Nicholas Genise and Daniele Micciancio. Faster gaussian sampling for trapdoor lattices with arbitrary modulus. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018*, pages 174–203, Cham, 2018. Springer International Publishing.

[19] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 197–206, 2008.

[20] Eu-Jin Goh, Stanislaw Jarecki, Jonathan Katz, and Nan Wang. Efficient signature schemes with tight reductions to the diffie-hellman problems. *J. Cryptol.*, 20(4):493–514, 2007.

[21] Dennis Hofheinz, Tibor Jager, Dakshita Khurana, Amit Sahai, Brent Waters, and Mark Zhandry. How to generate and use universal samplers. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part II*, volume 10032 of *Lecture Notes in Computer Science*, pages 715–744, 2016.

[22] James Howe, Thomas Prest, Thomas Ricosset, and Mélissa Rossi. Isochronous gaussian sampling: From inception to implementation. In Jintai Ding and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography*, pages 53–71, Cham, 2020. Springer International Publishing.

[23] Thomas Icart. How to hash into elliptic curves. In Shai Halevi, editor, *Advances in Cryptology - CRYPTO 2009, 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings*, volume 5677 of *Lecture Notes in Computer Science*, pages 303–316. Springer, 2009.

[24] Angshuman Karmakar, Sujoy Sinha Roy, Oscar Reparaz, Frederik Vercauteren, and Ingrid Verbauwhede. Constant-time discrete gaussian sampling. *IEEE Transactions on Computers*, 67(11):1561–1571, 2018.

[25] Charles F. F. Karney. Sampling exactly from the normal distribution. *ACM Transactions on Mathematical Software*, 42(1):1–14, Mar 2016.

[26] Allison B. Lewko and Brent Waters. Decentralizing attribute-based encryption. In Kenneth G. Paterson, editor, *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*, volume 6632 of *Lecture Notes in Computer Science*, pages 568–588. Springer, 2011.

[27] Vadim Lyubashevsky and Daniel Wichs. Simple lattice trapdoor sampling from a broad class of distributions. In Jonathan Katz, editor, *Public-Key Cryptography - PKC 2015 - 18th IACR International Conference on Practice and Theory in Public-Key Cryptography, Gaithersburg, MD, USA, March 30 - April 1, 2015, Proceedings*, volume 9020 of *Lecture Notes in Computer Science*, pages 716–730. Springer, 2015.

[28] Ueli Maurer, Renato Renner, and Clemens Holenstein. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In Moni Naor, editor, *Theory of Cryptography*, pages 21–39, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[29] Daniele Micciancio and Michael Walter. Gaussian sampling over the integers: Efficient, generic, constant-time. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part II*, volume 10402 of *Lecture Notes in Computer Science*, pages 455–485. Springer, 2017.

[30] Thomas Pöppelmann, Léo Ducas, and Tim Güneysu. Enhanced lattice-based signatures on reconfigurable hardware. In Lejla Batina and Matthew Robshaw, editors, *Cryptographic Hardware and Embedded Systems - CHES 2014 - 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings*, volume 8731 of *Lecture Notes in Computer Science*, pages 353–370. Springer, 2014.

[31] Thomas Ristenpart, Hovav Shacham, and Thomas Shrimpton. Careful with composition: Limitations of the indifferentiability framework. In *Proceedings of the 30th Annual International Conference on Theory and Applications of Cryptographic Techniques: Advances in Cryptology*, EUROCRYPT'11, page 487–506, Berlin, Heidelberg, 2011. Springer-Verlag.

[32] Raymond K. Zhao, Ron Steinfeld, and Amin Sakzad. Cosac: compact and scalable arbitrary-centered discrete gaussian sampling over integers. In Jintai Ding and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography*, Lecture Notes in Computer Science, pages 284–303. Springer, 2020. 11th International Conference on Post-Quantum Cryptography, PQCrypto 2020 ; Conference date: 15-04-2020 Through 17-04-2020.

[33] Raymond K. Zhao, Ron Steinfeld, and Amin Sakzad. Facct: Fast, compact, and constant-time discrete gaussian sampler over integers. *IEEE Transactions on Computers*, 69(1):126–137, 2020.

A Explaining Sampling over Small Ranges For Parameterized Distributions

To apply the result of Section 4 to SampleBase in Section 6, we can extend our definition of sampler to include parameterization of the distribution family on the same security parameter.

Definition 8. *Let f be a function from security parameter λ to a set of valid inputs. We say an algorithm* $\mathsf{SampleParam}(1^\lambda, \mathsf{p}; r)$ *is an $f(\lambda)$-parameterized sampler for a family of distributions $\{\mathcal{D}_{\mathsf{p},\lambda}\}_{\mathsf{p},\lambda}$ if* SampleParam *runs in* $\mathsf{poly}(\lambda)$ *time and for all $\mathsf{p} \in f(\lambda)$, the output of* $\mathsf{SampleParam}(1^\lambda, \mathsf{p}; r)$ *is statistically close to $\mathcal{D}_{\mathsf{p},\lambda}$.*

Definition 9. *Analogously, we say a* SampleParam *algorithm is explainable if there exists a (possibly randomized) algorithm* $\mathsf{ExplainParam}(1^\lambda, 1^\kappa, x, \mathsf{p})$ *such that* ExplainDG *runs in* $\mathsf{poly}(\lambda, \kappa)$ *time, and there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that for all $\mathsf{p} \in f(\lambda)$, the statistical distance between the following 2 distributions is at most $\frac{1}{\kappa} + \mathsf{negl}(\lambda)$.*

Game A

- $r \xleftarrow{R} \{0,1\}^n$.
- $x \leftarrow \mathsf{SampleParam}(1^\lambda, \mathsf{p}; r)$.
- Return r, x.

Game B

- $x \leftarrow \mathsf{SampleParam}(1^\lambda, \mathsf{p})$.
- $r \leftarrow \mathsf{ExplainParam}(1^\lambda, 1^\kappa, x, \mathsf{p})$.
- Return r, x.

Theorem 8. *Let* SampleParam *be a parameterized sampler for some distribution family $\{\mathcal{D}_\lambda\}_\lambda$ on range $\{R_{\mathsf{p},\lambda}\}_{\mathsf{p},\lambda}$. If $|R_{\mathsf{p},\lambda}| \leq \mathsf{poly}(\lambda)$ for all $\mathsf{p} \in f(\lambda)$, then* SampleParam *is explainable.*

Proof. First, we observe that

$\mathsf{Explain}(1^\lambda, 1^\kappa, x, \mathsf{p})$

- Repeat $\kappa \cdot |R_{\mathsf{p},\lambda}|$ times
  - Select fresh randomness r
  - $x' \leftarrow \mathsf{Sample}(1^\lambda, \mathsf{p}; r)$
  - If $x' = x$, stop and return r.
- Return $\perp$

Claim 27. Explain *runs in* $\mathsf{poly}(\lambda, \kappa)$ *time*

Proof. First, by the efficiency requirement of Definition 8 $\kappa \cdot |R_{\mathsf{p},\lambda}|$, SampleParam has runtime $\mathsf{poly}(\lambda)$. ExplainParam simply calls $\mathsf{SampleParam}(1^\lambda; r)$ up to$\kappa \cdot |R_{\mathsf{p},\lambda}|$ many times (along with some other minor efficient computation). Thus, we can bound the runtime with $\kappa \cdot |R_{\mathsf{p},\lambda}| \cdot \mathsf{poly}(\lambda)$. Since $|R_{\mathsf{p},\lambda}| \leq \mathsf{poly}(\lambda)$ by assumption, this bounds the runtime with $\kappa \cdot \mathsf{poly}(\lambda) \in \mathsf{poly}(\lambda, \kappa)$. □

Claim 28. ExplainParam *returns $\perp$ in Game B with probability $< \frac{1}{\kappa}$*

Proof. Observe that over the course of Game B, SampleParam is called on independent randomness up to $\kappa \cdot |R_{\mathsf{p},\lambda}| + 1$ times (once directly from the Game and $\kappa \cdot |R_{\mathsf{p},\lambda}|$ times in the execution of Game B). By pigeonhole, we know that among those calls, at most $|R_{\mathsf{p},\lambda}| - 1$ calls to SampleParam result in a unique x which is not returned by any of the other calls. Thus, since the calls are independent, the probability that the single call directly from Game B produced a unique x is at most $\frac{|R_{\mathsf{p},\lambda}|-1}{\kappa|R_{\mathsf{p},\lambda}|+1} < \frac{1}{\kappa}$. If this call is not unique, we can see ExplainParam finds $x' = x$ and does not return $\perp$. □

Claim 29. *The statistical distance of Game A and Game B in Definition 3 using* ExplainParam *is* $< \frac{1}{\kappa}$

Proof. We first note that if $\mathsf{ExplainParam}(1^\lambda, 1^\kappa, x, \mathsf{p})$ returned $r \neq \perp$, then $x = x' = \mathsf{Sample}(1^\lambda; r)$. Thus, since x is the same function of r in both games, it suffices to show that r in Game B has statistical distance $< \frac{1}{\kappa}$ to uniform.

Consider an alternate $\mathsf{ExplainParam}'$ which, rather than sampling x' at most $\kappa \cdot |R_\lambda|$ times, samples x' until it finds an appropriate r. Here, we can compute the probability that any particular r is chosen as the probability the x chosen for ExplainParam is equal to $\mathsf{SampleParam}(1^\lambda, param; r)$ multiplied by the probability that r is chosen conditional on $x' \leftarrow \mathsf{SampleParam}(1^\lambda, \mathsf{p}; r)$. This is equal to

$$\Pr[\mathsf{SampleParam}(1^\lambda, \mathsf{p}) = x] \cdot \frac{2^{-n}}{\Pr[\mathsf{SampleParam}(1^\lambda, \mathsf{p}) = x']} = 2^{-n}$$

so r is uniform on $\{0,1\}^n$. Now note that $\mathsf{ExplainParam}'$ only differs from ExplainParam when ExplainParam returns $\perp$ before finding such an r. By Claim 2, this happens with probability $< \frac{1}{\kappa}$, bounding the statistical distance. □

□

Corollary 9. *Let* SampleParam *be a sampler for some distribution family* $\{\mathcal{D}_{param,\lambda}\}_{param,\lambda}$. *If there exists sets* $\{S_{\mathsf{p},\lambda}\}_{param,\lambda}$ *where for all* $\mathsf{p} \in f(\lambda)$ $S_{\mathsf{p},\lambda} \subseteq R_{\mathsf{p},\lambda}$, $|S_{\mathsf{p},\lambda}| \leq \mathsf{poly}(\lambda)$, *and*

$$\Pr\left[\begin{matrix} x \xleftarrow{R} \mathcal{D}_{\mathsf{p},\lambda} \\ x \notin S_{\mathsf{p},\lambda} \end{matrix}\right] < \mathsf{negl}(\lambda)$$

, then Sample *is explainable.*

Proof. Let $\{\mathcal{D}'_{\mathsf{p},\lambda}\}_{\mathsf{p},\lambda}$ be the distribution $\{\mathcal{D}_{\mathsf{p},\lambda}\}_{\mathsf{p},\lambda}$ conditional on the output being $\in \{S_{\mathsf{p},\lambda}\}_{\mathsf{p},\lambda}$. It is easy to see that $\{\mathcal{D}'_{\mathsf{p},\lambda}\}_{\mathsf{p},\lambda}$ satisfies the conditions for Theorem 8, and so any sampler for $\{\mathcal{D}'_{\mathsf{p},\lambda}\}_{\mathsf{p},\lambda}$ is explainable. Since $\{\mathcal{D}'_{\mathsf{p},\lambda}\}_{\mathsf{p},\lambda}$ is statistically close to $\{\mathcal{D}_{\mathsf{p},\lambda}\}_{\mathsf{p},\lambda}$, any sampler for $\{\mathcal{D}_{param,\lambda}\}_{param,\lambda}$ is a sampler for $\{\mathcal{D}'_{\mathsf{p},\lambda}\}_{\mathsf{p},\lambda}$, and so is explainable. □

We remark that this definition is a generalization of the discrete Gaussian sampler in Definition 6.

B Rejection Sampling

We examine the rejection sampling method used in [19]. This is an application of a fairly general method of sampling in statistics. Here, a sample is drawn from the uniform distribution on a subset of $\mathbb{Z}$, and it is accepted with probability proportional to $\mathsf{ProbDens}(1^\lambda, x)$. A similar approach was sketched in [1]. We include a formal version here for completeness. We assume without loss of generality that the center c of the discrete gaussian is between $[0, 1)$, as otherwise, it can be offset by an integer amount.

$\mathsf{SampleDG}(1^\lambda, c, \sigma; r = \{(x_i \in [-\lambda\sigma, \lambda\sigma]_\mathbb{Z}, p_i \in [0,1])\}_{i \in \lambda^2})$

- For $i \in [\lambda^2]_\mathbb{Z}$:
  - Let $x'_i = x_i - c$.
  - Return x'_i if $p_i \leq exp\left(-\frac{(x'_i)^2}{2\sigma^2}\right)$.
- Return $\perp$

Lemma 15. SampleDG *is an* $\exp(\lambda)$*-discrete Gaussian sampler.*

See [19] for proof.

Lemma 16. SampleDG *is an* ***explainable*** $\exp(\lambda)$*-discrete Gaussian sampler.*

Proof. Consider the following ExplainDG algorithm

$\mathsf{ExplainDG}(1^\lambda, 1^\kappa, x, c, \sigma)$

- For $i \in [\lambda^2]_\mathbb{Z}$, sample $x_i \xleftarrow{R} [-\lambda\sigma, +\lambda\sigma]_\mathbb{Z}$ and $p_i \in [0,1]$.
- Initialize list of randomness $r = \{(x_i, p_i)\}_{i \in \lambda^2}$
- For $i \in [\lambda^2]_\mathbb{Z}$:
  - Let $x_i' = x_i - c$.
  - If $p_i \leq exp\left(-\frac{(x_i')^2}{2\sigma^2}\right)$
    - Replace x_i' by setting $x_i' = x - c$
    - Sample $p_i \xleftarrow{R} [0, exp\left(-\frac{x_i^2}{2\sigma^2}\right)]$
    - Return r
- Return $\perp$.

Claim 30. ExplainDG *runs in* $\mathsf{poly}(\lambda, \kappa)$ *time*

Proof. ExplainDG simply generates a polynomial amount of randomness (λ^2 items with an integer from an exponential domain and a number from $[0,1]$) and iterates through it once, doing some efficient arithmetic computation. □

Claim 31. ExplainDG *returns* $\perp$ *with probability* $\mathsf{negl}(\lambda)$.

Proof. We observe that ExplainDG only returns $\perp$ when all $p_i > exp\left(-\frac{(x_i-c)^2}{2\sigma^2}\right)$, the same condition for which SampleDG returns $\perp$. For the same reasoning as in [19], this returns $\perp$ with negligible probability. □

Claim 32. *The statistical distance of Game A and Game B in Definition 3 using* ExplainDG *is* $< \frac{1}{\kappa} + \mathsf{negl}(\lambda)$

Proof. First, we can observe that by construction, $\mathsf{ExplainDG}(1^\lambda, 1^\kappa, c, \sigma; r)$ only returns r such that $x = \mathsf{SampleDG}(1^\lambda, c, \sigma; r)$ or $\perp$. Thus, since x is the same function of r in both Game A and B, it suffices to show that the distribution of r (and not x) in Game B is statistically close to uniform.

Consider some randomness $r^* = \{(x_i, p_i)\}_{i \in [\lambda^2]}$. We break it down into two cases - either $\mathsf{SampleDG}(1^\lambda, c, \sigma; r^*)$ returns $\perp$ or $\mathsf{SampleDG}(1^\lambda, c, \sigma; r^*) \neq \perp$.

In the former case, from Claim 31, we know that this occurs with probability $\mathsf{negl}(\lambda)$, and thus represents a negligible fraction of the statistical weight.

In the latter case, we know there is some i^* for which SampleDG returns x_{i^*} on. From the definition of SampleDG, we can see this only happens when $p_{i**} \leq exp\left(-\frac{(x_{i^*}-c)^2}{2\sigma^2}\right)$. Since each sample (x_i, p_i) is generated independently, the distribution of i^* is geometric, with $i^* = k$ with probability $p(1-p)^{k-1}$, where p is the probability $p_i \leq exp\left(-\frac{(x_i-c)^2}{2\sigma^2}\right)$ for a random pair (x_i, p_i). We can see the probability that ExplainDG returns an r with that particular i^* is $p(1-p)^{i^*-1}$, since it generates independent random pairs, and only modifies the first one which accepts. Next, we can observe that the for ExplainDG to generate that particular r^*, it needs to be given $x = x_{i^*}$ as input, then pick the exact p_{i^*} value. Since the input x is sampled from SampleDG, we know this occurs with $\Pr[x' \xleftarrow{R} \{\mathcal{D}_\lambda\}_\lambda \;\; x = x']$. Next, the probability the exact p_{i^*} is picked is inversely proportional to the interval length, $exp\left(-\frac{(x_{i^*}-c)^2}{2\sigma^2}\right)$, so the probability that r is chosen is proportional to

$$\frac{\Pr[x' \xleftarrow{R} \mathcal{D}_\lambda \;\; x_{i^*} = x']}{exp\left(-\frac{(x_{i^*}-c)^2}{2\sigma^2}\right)}$$

However, recall from the definition of discrete Gaussian that

$$\frac{\Pr[x' \xleftarrow{R} \mathcal{D}_\lambda \;\; x_{i^*} = x']}{exp\left(-\frac{(x_{i^*}-c)^2}{2\sigma^2}\right)} = \frac{exp\left(-\frac{(x_{i^*}-c)^2}{2\sigma^2}\right)}{\sum_{x\in\mathbb{Z}} e^{-\frac{(x-c)^2}{2\sigma^2}}} \cdot \frac{1}{exp\left(-\frac{(x_{i^*}-c)^2}{2\sigma^2}\right)} = \frac{1}{\sum_{x\in\mathbb{Z}} e^{-\frac{(x-c)^2}{2\sigma^2}}}.$$

This is a constant, so we can conclude that the probability of any particular r^* being returned in this case is independent of r^*, and hence uniform.

□

Combining Claim 30 and Claim 32, we get that SampleDG is explainable.

□