Isochronous Gaussian Sampling: From Inception to Implementation

With Applications to the Falcon Signature Scheme

James Howe[1], Thomas Prest[1], Thomas Ricosset[2], and Mélissa Rossi[2,3,4,5]

[1] PQShield, Oxford, UK
james.howe@pqshield.com thomas.prest@pqshield.com
[2] Thales, Gennevilliers, France
thomas.ricosset@thalesgroup.com
[3] ANSSI, Paris, France
[4] École normale supérieure, CNRS, PSL University, Paris, France
[5] Inria, Paris, France
melissa.rossi@ens.fr

Abstract Gaussian sampling over the integers is a crucial tool in lattice-based cryptography, but has proven over the recent years to be surprisingly challenging to perform in a generic, efficient and provable secure manner. In this work, we present a modular framework for generating discrete Gaussians with arbitrary center and standard deviation. Our framework is extremely simple, and it is precisely this simplicity that allowed us to make it easy to implement, provably secure, portable, efficient, and provably resistant against timing attacks. Our sampler is a good candidate for any trapdoor sampling and it is actually the one that has been recently implemented in the Falcon signature scheme. Our second contribution aims at systematizing the detection of implementation errors in Gaussian samplers. We provide a statistical testing suite for discrete Gaussians called SAGA (Statistically Acceptable GAussian). In a nutshell, our two contributions take a step towards trustable and robust Gaussian sampling real-world implementations.

Keywords.

Lattice based cryptography, Gaussian Sampling, Isochrony, Statistical verification tools

1 Introduction

Gaussian sampling over the integers is a central building block of lattice-based cryptography, in theory as well as in practice. It is also notoriously difficult to perform efficiently and securely, as illustrated by numerous side-channel attacks exploiting BLISS' Gaussian sampler [9,23,54,61]. For this reason, some schemes limit or proscribe the use of Gaussians [6,40]. However, in some situations, Gaussians are unavoidable. The most prominent example is trapdoor sampling [30,53,45]: performing it with other distributions is an open question, except in limited cases [41] which entail a growth $O(\sqrt{n})$ to $O(n)$ of the output, resulting in dwindling security levels. Given the countless applications of trapdoor sampling (full-domain hash signatures [30,58], identity-based encryption (or IBE) [30,20], hierarchical IBE [11,1], etc.), it is important to come up with Gaussian samplers over the integers which are not only efficient, but also provably secure, resistant to timing attacks, and in general easy to deploy.

Our first contribution is to propose a Gaussian sampler over the integers with all the properties which are expected of a sampler for widespread deployment. It is simple and modular, making analysis and subsequent improvements easy. It is efficient and portable, making it amenable to a variety of scenarios. Finally, we formally prove its security and resistance against timing attacks. We detail below different aspects of our sampler:

- **Simplicity and Modularity.** At a high level, our framework only requires two ingredients (a base sampler and a rejection sampler) and combines them in a simple and black-box way. Not only does it make the description of our sampler modular (as one can replace any of the ingredients), this simplicity and modularity also infuses all aspects of its analysis.
- **Genericity.** Our sampler is fully generic as it works with arbitrary center μ and standard deviation σ. In addition, it does not incur hidden precomputation costs: given a fixed base sampler of parameter $\sigma_{\max}$, our framework allows to sample from $D_{\mathbb{Z},\sigma,\mu}$ for any $\eta_\epsilon(\mathbb{Z}^n) \leq \sigma \leq \sigma_{\max}$. In comparison, [47] implicity requires a different base sampler for each different value of σ; this limits its applicability for use cases such as Falcon [58], which has up to 2048 different σ's, all computed at key generation.
- **Efficiency and Portability.** Our sampler is instantiated with competitive parameters which make it very efficient in time and memory usage. For $\sigma_{\max} = 1.8205$ and SHAKE256 used as PRNG, our sampler uses only 512 bytes of memory and achieved 1,848,428 samples per second on an Intel i7-6500U clocked at 2.5 GHz. Moreover, our sampler can be instantiated in a way that uses only integer operations, making it highly portable.
- **Provable Security.** A security analysis based on the statistical distance would either provide very weak security guarantees or require to increase the running time by an order of magnitude. We instead rely on the Rényi divergence, a tool which in the recent years has allowed dramatic efficiency gains for lattice-based schemes [3,57]. We carefully selected our parameters as to make them as amenable to a Rényi divergence-based analysis.
- **Isochrony.** We formally show that our sampler is isochronous: its running time is independent of the inputs σ, μ and of the output z. Isochrony is weaker than being constant-time, but it nevertheless suffices to argue security against timing attacks. Interestingly, our proof of isochrony relies on techniques and notions that are common in lattice-based cryptography: the smoothing parameter, the Rényi divergence, etc. In particular, the isochrony of our sampler is implied by parameters dictated by the current state of the art for *black-box* security of lattice-based schemes.

One second contribution stems from a simple observation: implementations of otherwise perfectly secure schemes have failed in spectacular ways by introducing weaknesses, a common one being randomness failure: this is epitomized by nonce reuses in ECDSA, leading to jailbreaking Sony PS3 consoles[1] and exposing Bitcoin wallets [8]. The post-quantum community is aware of this point of failure but does not seem to have converged on a systematic way to mitigate it [51]. Randomness failures have been manually discovered and fixed in implementations of Dilithium [50], Falcon [56,52] and other schemes; the case of Falcon is particularly relevant to us because the sampler implemented was the one described in this document!

Our second contribution is a first step at systematically detecting such failures: we propose a statistical test suite called SAGA for validating discrete Gaussians. This test suite can check univariate samples; we therefore use it to validate our own implementation of our sampler. In addition, our test suite can check multivariate Gaussians as well; this enables validation at a higher level: if the base sampler over the integers is validated, but the output of the high-level scheme does not behave like a multivariate Gaussian even though the theory predicts it should, then this is indicative of an implementation mistake somewhere else in the implementation (or, at the worst case, that the theory is deficient). We illustrate that with a simple example of a (purportedly) deficient implementation of Falcon [58], however it can be used for any other scheme sampling multivariate discrete Gaussians, including but not limited to [45,20,29,5,12]. The test suite is publicly available at: https://github.com/PQShield/SAGA.

[1] https://media.ccc.de/v/27c3-4087-en-console_hacking_2010.

2 Related Works

In the recent years, there has been a surge of works related to Gaussian sampling over the integers. Building on convolution techniques from [55], [47] proposed an arbitrary-center Gaussian sampler base, as well as a statistical tool (the max-log distance) to analyse it. [3,57,44] revisited classical techniques with the Rényi divergence. Polynomial-based methods were further studied by [57,65,4]. The use of rounded Gaussians was proposed in [34]. Knuth-Yao's DDG trees have been considered in [22,35].[2] Lazy floating-point precision was studied in [21,18]. We note that techniques dating back to von Neumann [62] allow to generate (continuous) Gaussians elegantly using finite automata [27,2,36]. While these have been considered in the context of lattice-based cryptography [19,17] they are also notoriously hard to make isochronous. Finally, [63] studied previously cited techniques with the goal of minimizing their relative error.

3 Preliminaries

3.1 Gaussians

For $\sigma, \mu \in \mathbb{R}$ with $\sigma > 0$, we call Gaussian function of parameters σ, μ and denote by $\rho_{\sigma,\mu}$ the function defined over $\mathbb{R}$ as $\rho_{\sigma,\mu}(x) = \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$. Note that when $\mu = 0$ we omit it in index notation, e.g. $\rho_\sigma(x) = \rho_{\sigma,0}(x)$. The parameter σ (resp. μ) is often called the standard deviation (resp. center) of the Gaussian. In addition, for any countable set $S \subsetneq \mathbb{R}$ we abusively denote by $\rho_{\sigma,\mu}(S)$ the sum $\sum_{z\in S} \rho_{\sigma,\mu}(z)$. When $\sum_{z\in S} \rho_{\sigma,\mu}(z)$ is finite, we denote by $D_{S,\sigma,\mu}$ and call Gaussian distribution of parameters σ, μ the distribution over S defined by $D_{S,\sigma,\mu}(z) = \rho_{\sigma,\mu}(z)/\rho_{\sigma,\mu}(S)$. Here too, when $\mu = 0$ we omit it in index notation, e.g. $D_{S,\sigma,\mu}(z) = D_{S,\sigma}(z)$. We use the notation $\mathcal{B}_p$ to denote the Bernoulli distribution of parameter p.

3.2 Renyi Divergence

We recall the definition of the Rényi divergence, which we will use massively in our security proofs.

Definition 1 (Rényi Divergence). *Let $\mathcal{P}$, $\mathcal{Q}$ be two distributions such that $Supp(\mathcal{P}) \subseteq Supp(\mathcal{Q})$. For $a \in (1, +\infty)$, we define the Rényi divergence of order a by*

$$R_a(\mathcal{P}, \mathcal{Q}) = \left(\sum_{x \in Supp(\mathcal{P})} \frac{\mathcal{P}(x)^a}{\mathcal{Q}(x)^{a-1}} \right)^{\frac{1}{a-1}}.$$

In addition, we define the Rényi divergence of order $+\infty$ by

$$R_\infty(\mathcal{P}, \mathcal{Q}) = \max_{x \in Supp(\mathcal{P})} \frac{\mathcal{P}(x)}{\mathcal{Q}(x)}.$$

The Rényi divergence is not a distance; for example, it is neither symmetric nor does it verify the triangle inequality, which makes it less convenient than the statistical distance. On the other hand, it does verify cryptographically useful properties, incluing a few listed below.

Lemma 2 ([3]). *For two distributions $\mathcal{P}, \mathcal{Q}$ and two families of distributions $(\mathcal{P}_i)_i, (\mathcal{Q}_i)_i$, the Rényi divergence verifies these properties:*

[2] We note that one could use [35] to speed up our base sampler; however this results in a huge code size (more than 50kB). Since the running time of the base sampler was not a bottleneck for the usecase we considered, we instead relied on a straightforward, slightly less efficient CDT-based method.

- ***Data processing inequality.*** *For any function f, $R_a(f(\mathcal{P}), f(\mathcal{Q})) \leq R_a(\mathcal{P}, \mathcal{Q})$.*
- ***Multiplicativity.*** $R_a(\prod_i \mathcal{P}_i, \prod_i \mathcal{Q}_i) = \prod_i R_a(\mathcal{P}_i, \mathcal{Q}_i)$.
- ***Probability preservation.*** *For any event $E \subseteq \mathrm{Supp}(\mathcal{Q})$ and $a \in (1, +\infty)$,*

$$\mathcal{Q}(E) \geq \mathcal{P}(E)^{\frac{a}{a-1}} / R_a(\mathcal{P}, \mathcal{Q}), \tag{1}$$
$$\mathcal{Q}(E) \geq \mathcal{P}(E) / R_\infty(\mathcal{P}, \mathcal{Q}). \tag{2}$$

The following lemma shows that a bound of δ on the relative error between two distributions implies a bound $O(a\delta^2)$ on the log of the Rényi divergence (as opposed to a bound $O(\delta)$ on the statistical distance).

Lemma 3 (Lemma 3 of [57]). *Let $\mathcal{P}, \mathcal{Q}$ be two distributions of same support Ω. Suppose that the relative error between $\mathcal{P}$ and $\mathcal{Q}$ is bounded: $\exists \delta > 0$ such that $\left|\frac{\mathcal{P}}{\mathcal{Q}} - 1\right| \leq \delta$ over Ω. Then, for $a \in (1, +\infty)$:*

$$R_a(\mathcal{P}, \mathcal{Q}) \leq \left(1 + \frac{a(a-1)\delta^2}{2(1-\delta)^{a+1}}\right)^{\frac{1}{a-1}} \underset{\delta \to 0}{\sim} 1 + \frac{a\delta^2}{2}$$

3.3 Smoothing Parameter

For $\epsilon > 0$, the smoothing parameter $\eta_\epsilon(\Lambda)$ of a lattice Λ is the smallest value $\sigma > 0$ such that $\rho_{\frac{1}{\sigma\sqrt{2\pi}}}(\Lambda^\star \backslash \{\mathbf{0}\}) \leq \epsilon$, where $\Lambda^\star$ denotes the dual of Λ. In the literature, some definitions of the smoothing parameter scale our definition by a factor $\sqrt{2\pi}$. It is shown in [46] that $\eta_\epsilon(\mathbb{Z}^n) \leq \eta_\epsilon^+(\mathbb{Z}^n)$, where:

$$\eta_\epsilon^+(\mathbb{Z}^n) = \frac{1}{\pi}\sqrt{\frac{1}{2}\log\left(2n\left(1 + \frac{1}{\epsilon}\right)\right)}. \tag{3}$$

3.4 Isochronous algorithms

We now give a semi-formal definition of isochronous algorithms.

Definition 4. *Let $\mathcal{A}$ be a (probabilistic or deterministic) algorithm with set of input variables $\mathcal{I}$, set of output variables $\mathcal{O}$, and let $\mathcal{S} \subseteq \mathcal{I} \cup \mathcal{O}$ be the set of sensitive variables. We say that $\mathcal{A}$ is perfectly isochronous with respect to $\mathcal{S}$ if its running time is independent of any variable in $\mathcal{S}$.*

In addition, we say that $\mathcal{A}$ statistically isochronous with respect to $\mathcal{S}$ if there exists a distribution $\mathcal{D}$ independent of all the variables in $\mathcal{S}$, such that the running time of $\mathcal{A}$ is statistically close (for a clearly identified divergence) to $\mathcal{D}$.

We note that we can define a notion of computationally isochronous algorithm. For such an algorithm, it is computationally it hard to recover the sensitive variables even given the distribution of the running time of the algorithm. We can even come up with a contrived example of such an algorithm: let $\mathcal{A}()$ select in an isochronous manner an x uniformly in a space of min-entropy $\geq \lambda$, compute $y = H(x)$ and wait a time y before outputting x. One can show that recovering x given the running time of $\mathcal{A}$ is hard if H is a one-way function.

4 The sampler

In this section, we describe our new sampler with arbitrary standard deviation and center. The main assumption of our setting is to consider that all the standard deviations are bounded and that the center is in $[0, 1]$. In other words, denoting the upper bound and lower bound on the

Algorithm 1 $\mathsf{SamplerZ}(\sigma, \mu)$

Require: $\mu \in [0,1]$, $\sigma \leq \sigma_{\max}$
Ensure: $z \sim D_{\mathbb{Z},\sigma,\mu}$
1: **while** True **do**
2: $z_0 \leftarrow \mathsf{BaseSampler}()$
3: $b \leftarrow \{0,1\}$ uniformly
4: $z \coloneqq (2b-1) \cdot z_0 + b$
5: $x \coloneqq \frac{z_0^2}{2\sigma_{\max}^2} - \frac{(z-\mu)^2}{2\sigma^2}$
6: **if** $\mathsf{AcceptSample}(\sigma, x)$ **then**
7: **return** z

Algorithm 2 $\mathsf{AcceptSample}(\sigma, x)$

Require: $\sigma_{\min} \leq \sigma \leq \sigma_{\max}, x < 0$
Ensure: $b \sim \mathcal{B}_{\frac{\sigma_{\min}}{\sigma} \cdot \exp(x)}$
1: $p \coloneqq \frac{\sigma_{\min}}{\sigma} \cdot \mathsf{ApproxExp}(x)$
Lazy Bernoulli sampling
2: $i \coloneqq 1$
3: **do**
4: $i \coloneqq i \cdot 2^8$
5: $u \leftarrow [\![0, 2^8 - 1]\!]$ uniformly
6: $v \coloneqq \lfloor p \cdot i \rfloor$ `& 0xff`
7: **while** $u = v$
8: **return** $(u < v)$

standard deviation as $\sigma_{\max} > \sigma_{\min} > 0$, we present an algorithm that samples the distribution $D_{\mathbb{Z},\sigma,\mu}$ for any $\mu \in [0,1]$ and $\sigma_{\min} \leq \sigma \leq \sigma_{\max}$.

Our sampling algorithm is called SamplerZ and is described in Algorithm 1. We denote by BaseSampler an algorithm that samples an element with the fixed half Gaussian distribution $D_{\mathbb{Z}^+,\sigma_{\max}}$. The first step consists in using BaseSampler. The obtained z_0 sample is then transformed into $z \coloneqq (2b-1) \cdot z_0 + b$ where b is a bit drawn uniformly in $\{0,1\}$. Let us denote by $BG_{\sigma_{\max}}$ the distribution of z. The distribution of z is a discrete bimodal half-Gaussian of centers 0 and 1. More formally,

$$BG_{\sigma_{\max}}(z) = \frac{1}{2} \begin{cases} D_{\mathbb{Z}^+,\sigma_{\max}}(-z) & \text{if } z \leq 0 \\ D_{\mathbb{Z}^+,\sigma_{\max}}(z-1) & \text{if } z \geq 1. \end{cases} \tag{4}$$

Then, to recover the desired distribution $D_{\mathbb{Z},\sigma,\mu}$ for the inputs (σ, μ), one might want to apply the classical rejection sampling technique applied to lattice based schemes [39] and accept z with probability

$$\begin{aligned} \frac{D_{\mathbb{Z},\sigma,\mu}(z)}{BG_{\sigma_{\max}}(z)} &= \begin{cases} \exp\left(\frac{z^2}{2\sigma_{\max}^2} - \frac{(z-\mu)^2}{2\sigma^2}\right) & \text{if } z \leq 0 \\ \exp\left(\frac{(z-1)^2}{2\sigma_{\max}^2} - \frac{(z-\mu)^2}{2\sigma^2}\right) & \text{if } z \geq 1 \end{cases} \\ &= \exp\left(\frac{z_0^2}{2\sigma_{\max}^2} - \frac{(z-\mu)^2}{2\sigma^2}\right). \end{aligned}$$

The element inside the exp is computed in step 5. Next, we also introduce an algorithm denoted AcceptSample. The latter performs the rejection sampling (Algorithm 2): using ApproxExp an algorithm that returns $\exp(\cdot)$, it returns a Bernoulli sample with the according probability. Actually, for isochrony matters, detailed in Section 6, the latter acceptance probability is rescaled by a factor $\frac{\sigma_{\min}}{\sigma}$. As z follows the $BG_{\sigma_{\max}}$ distribution, after the rejection sampling, the final distribution of $\mathsf{SamplerZ}(\sigma, \mu)$ is then proportional to $\frac{\sigma_{\min}}{\sigma} \cdot D_{\mathbb{Z},\sigma,\mu}$ which is, after normalization exaclty equal to $D_{\mathbb{Z},\sigma,\mu}$. Thus, with this construction, one can derive the following proposition.

Proposition 5 (Correctness). *Assume that all the uniform distributions are perfect and that* $\mathsf{BaseSampler} = D_{\mathbb{Z}^+,\sigma_{\max}}$ *and* $\mathsf{ApproxExp} = \exp$*, then the construction of* SamplerZ *(in Algorithms 1 and 2) is such that* $\mathsf{SamplerZ}(\sigma, \mu) = D_{\mathbb{Z},\sigma,\mu}$.

In practical implementations, one cannot acheive perfect distributions. Only acheiving $\mathsf{BaseSampler} \approx D_{\mathbb{Z}^+,\sigma_{\max}}$ and $\mathsf{ApproxExp} \approx \exp$ is possible.Section 6 proves that, under certain conditions on BaseSampler and ApproxExp and on the number of sampling queries, the final distribution remains indistinguishable from $D_{\mathbb{Z},\sigma,\mu}$.

Table 1: Number of calls to SamplerZ, BaseSampler and ApproxExp

	Notation	Value for Falcon
Calls to sign (as per NIST)	Q_{s}	$\leq 2^{64}$
Calls to SamplerZ	Q_{samplZ}	$Q_{\mathrm{s}} \cdot 2 \cdot n \leq 2^{75}$
Calls to BaseSampler	Q_{bs}	$\mathcal{N}_{\mathrm{iter}} \cdot Q_{\mathrm{samplZ}} \leq 2^{76}$
Calls to ApproxExp	Q_{exp}	$Q_{\mathrm{bs}} \leq 2^{76}$

5 Proof of Security

Table 1 gives the notations for the number of calls to SamplerZ, BaseSampler and ApproxExp and the considered values when the sampler is instanciated for Falcon. Due to the rejection sampling in step 6, there will be a (potentially infinite) number of iterations of the **while** loop. We will show later in Lemma 7, that the number of iterations follows a geometric law of parameter $\approx \frac{\sigma_{\min} \cdot \sqrt{2\pi}}{2 \cdot \rho_{\sigma_{\max}}(\mathbb{Z}^+)}$. We note $\mathcal{N}_{\mathrm{iter}}$ a heuristic considered maximum number of iterations. By a central limit argument, $\mathcal{N}_{\mathrm{iter}}$ will only be marginally higher than the expected number of iterations. To instantiate the values $Q_{\mathrm{exp}} = Q_{\mathrm{bs}} = \mathcal{N}_{\mathrm{iter}} \cdot Q_{\mathrm{samplZ}}$ for the example of Falcon, we take $\mathcal{N}_{\mathrm{iter}} = 2$. In fact, $\frac{\sigma_{\min} \cdot \sqrt{2\pi}}{2 \cdot \rho_{\sigma_{\max}}(\mathbb{Z}^+)} \leq 2$ for Falcon's parameters.

The following Theorem estimates the security of SamplerZ, it is independant of the chosen values for the number of calls.

Theorem 6 (Security of SamplerZ). *Let λ_{IDEAL} (resp. λ_{REAL}) be the security parameter of an implementation using the perfect distribution $D_{\mathbb{Z},\sigma,\mu}$ (resp. the real distribution SamplerZ). If both following conditions are respected, at most two bits of security are lost. In other words, $\Delta\lambda \coloneqq \lambda_{\mathrm{IDEAL}} - \lambda_{\mathrm{REAL}} \leq 2$.*

$$\forall x < 0, \quad \left| \frac{\mathit{ApproxExp}(x) - \exp(x)}{\exp(x)} \right| \leq \sqrt{\frac{2 \cdot \lambda_{\mathrm{REAL}}}{2 \cdot (2 \cdot \lambda_{\mathrm{REAL}} + 1)^2 \cdot Q_{\mathrm{exp}}}} \qquad \text{(Cond. (1))}$$

$$R_{2 \cdot \lambda_{\mathrm{REAL}}+1}\left(\mathit{BaseSampler}, D_{\mathbb{Z}^+,\sigma_{\max}}\right) \leq 1 + \frac{1}{4 \cdot Q_{\mathrm{bs}}} \qquad \text{(Cond. (2))}$$

The proof of this Theorem is given in Appendix A.

To get concrete numerical values, we assume that 256 bits are claimed on the original scheme, thus 254 bits of security are claimed for the real implementation. Then for an implementation of Falcon, the numerical values are

$$\sqrt{\frac{2 \cdot \lambda_{\mathrm{REAL}}}{2 \cdot (2 \cdot \lambda_{\mathrm{REAL}} + 1)^2 \cdot Q_{\mathrm{exp}}}} \approx 2^{-43} \quad \text{and} \quad \frac{1}{4 \cdot Q_{\mathrm{bs}}} \approx 2^{-78}.$$

5.1 Instanciating the ApproxExp

To achieve condition (1) with ApproxExp, we use a polynomial approximation of the exponential on $[-\ln(2), 0]$. In fact, one can reduce the parameter x modulo $\ln(2)$ such that $x = -r - s\ln(2)$. Compute the exponential remains to compute $\exp(x) = 2^{-s}\exp(-r)$. Noting that $s \geq 64$ happen very rarely, thus s can be saturated at 63 to avoid overflow without loss in precision.

We use the polynomial approximation tool provided in GALACTICS [4]. This tool generates polynomial approximations that allow a computation in fixed precision with chosen size of coefficients and degree. As an example, for 32-bit coefficients and a degree 10, we obtain a polynomial $P_{\mathrm{gal}}(x) \coloneqq \sum_{i=0}^{10} a_i \cdot x^i$, with:

- $a_0 = 1$;
- $a_1 = 1$;
- $a_2 = 2^{-1}$;
- $a_3 = 2863311530 \cdot 2^{-34}$;
- $a_4 = 2863311481 \cdot 2^{-36}$;
- $a_5 = 2290647631 \cdot 2^{-38}$;
- $a_6 = 3054141714 \cdot 2^{-41}$;
- $a_7 = 3489252544 \cdot 2^{-44}$;
- $a_8 = 3473028713 \cdot 2^{-47}$;
- $a_9 = 2952269371 \cdot 2^{-50}$;
- $a_{10} = 3466184740 \cdot 2^{-54}$.

For any $x \in [-\ln(2), 0]$, P_{gal} verifies $\left|\frac{P_{\text{gal}}(x) - \exp(x)}{\exp(x)}\right| \leq 2^{-47}$, which is sufficient to verify condition (1) for Falcon implementation.

Flexibility on the implementation of the polynomial. Depending on the platform and the requirement for the signature, one can adapt the polynomial to fit their constraints. For example, if one wants to minimize the number of multiplications, implementing the polynomial with Horner's form is the best option. The polynomial is written in the following form :

$$P_{\text{gal}}(x) = a_0 + x(a_1 + x(a_2 + x(a_3 + x(a_4 + x(a_5 + x(a_6 + x(a_7 + x(a_8 + x(a_9 + xa_{10}))))))))).$$

Evaluating P_{gal} is then done serially as follows:

$$\begin{aligned}
&\mathtt{y} \leftarrow \mathtt{a_{10}} \\
&\mathtt{y} \leftarrow \mathtt{a_9} + \mathtt{y} \times \mathtt{x} \\
&\quad\vdots \\
&\mathtt{y} \leftarrow \mathtt{a_1} + \mathtt{y} \times \mathtt{x} \\
&\mathtt{y} \leftarrow \mathtt{a_0} + \mathtt{y} \times \mathtt{x}
\end{aligned}$$

Some architectures with small register sizes may be faster if the size of the coefficients of the polynomial is minimized, thus GALACTICS tool can be used to generate a polynomial with smaller coefficients. For example, we propose an alternative polynomial approximation on $[0, \frac{ln(2)}{64}]$ with 25 bits coefficients.

$$P = 1 + x + 2^{-1}x^2 + 699051 \cdot 2^{-22} \cdot x^3 + 699299 \cdot 2^{-24} \cdot x^4 + 605552 \cdot 2^{-26} \cdot x^5$$

To recover the polynomial approximation on $[0, ln(2)]$, we compute $P(\frac{x}{64})^{64}$.

Some architectures enjoy some level of parallelism, in which case it is desirable to minimise the depth of the circuit computing the polynomial[3]. Writing P_{gal} in Estrin's form [24] is helpful in this regard.

$$\begin{aligned}
&\mathtt{x_2} \quad \leftarrow \mathtt{x} \times \mathtt{x} \\
&\mathtt{x_4} \quad \leftarrow \mathtt{x_2} \times \mathtt{x_2} \\
&\mathtt{P_{gal}(x)} \leftarrow (\mathtt{x_4} \times \mathtt{x_4}) \times ((\mathtt{a_8} + \mathtt{a_9} \times \mathtt{x}) + \mathtt{x_2} \times \mathtt{a_{10}}) \\
&\qquad + (((\mathtt{a_0} + \mathtt{a_1} \times \mathtt{x}) + \mathtt{x_2} \times (\mathtt{a_2} + \mathtt{a_3} \times \mathtt{x})) + \mathtt{x_4} \times ((\mathtt{a_4} + \mathtt{a_5} \times \mathtt{x}) + \mathtt{x_2} \times (\mathtt{a_6} + \mathtt{a_7} \times \mathtt{x})))
\end{aligned}$$

5.2 Instanciating the BaseSampler

To achieve condition (2) with BaseSampler, we rely on a cumulative distribution table (CDT). We precompute a table of the cumulative distribution function of $D_{\mathbb{Z}^+,\sigma_{\max}}$ with a certain precision; then, to produce a sample, we generate a random value in $[0, 1]$ with the same precision, and return the index of the last entry in the table that is greater than that value. In variable time, the sampling can be done rather efficiently with a binary search, but a constant-time implementation has essentially no choice but to read the entire table each time and carry out each comparison. This process is summed up in Algorithm 3. The parameters w and θ are respectively the number

Algorithm 3 SampleCDT: full-table access CDT

$z \leftarrow 0$
$u \leftarrow [0, 1)$ uniformly with θ bits of absolute precision
for $0 \leq i \leq w$ **do**
  $b \leftarrow (\text{CDT}[w] \geq u)$ ▷ $b = 1$ if it is true and 0 otherwise
  $z \leftarrow z + b$
return z

of elements of the CDT and the precision of its coefficients. Let $a = 2 \cdot \lambda_{\text{REAL}} + 1$. To derive the parameters w and θ we use a simple script that, given $\sigma_{\max}$ and θ as inputs:

1. Computes the smallest tailcut w such that the Renyi divergence R_a between the ideal distribution $D_{\mathbb{Z}^+,\sigma_{\max}}$ and its restriction to $\{0, \ldots, w\}$ (noted $D_{[w],\sigma_{\max}}$) verifies $R_a(D_{[w],\sigma_{\max}}, D_{\mathbb{Z}^+,\sigma_{\max}}) \leq 1 + 1/(4Q_{\text{bs}})$;
2. Rounds the probability density table (PDT) of $D_{[w],\sigma_{\max}}$ with θ bits of absolute precision. This rounding is done "cleverly" by truncating all the PDT values except the largest:
   - for $z \geq 1$, the value $D_{[w],\sigma_{\max}}(z)$ is truncated: $PDT(z) = 2^{-\theta} \left\lfloor 2^{\theta} D_{[w],\sigma_{\max}}(z) \right\rfloor$.
   - in order to have a probability distribution, $PDT(0) = 1 - \sum_{z \geq 1} PDT(z)$.
3. Derives the CDT from the PDT and computes the final $R_a(\mathsf{SampleCDT}_{w=19,\theta=72}, D_{\mathbb{Z}^+,\sigma_{\max}})$.

Taking $\sigma_{\max} = 1.8205$ and $\theta = 72$ as inputs, we found $w = 19$.

- $\texttt{PDT}(0) = 2^{-72} \times 16976802417466403000 30$
- $\texttt{PDT}(1) = 2^{-72} \times 14599434566429129596 16$
- $\texttt{PDT}(2) = 2^{-72} \times 92848835501801105651 5$
- $\texttt{PDT}(3) = 2^{-72} \times 43669394481705441461 9$
- $\texttt{PDT}(4) = 2^{-72} \times 15189314079036920101 3$
- $\texttt{PDT}(5) = 2^{-72} \times 39071441848292237840$
- $\texttt{PDT}(6) = 2^{-72} \times 7432604049020375675$
- $\texttt{PDT}(7) = 2^{-72} \times 1045641569992574730$
- $\texttt{PDT}(8) = 2^{-72} \times 108788995549429682$
- $\texttt{PDT}(9) = 2^{-72} \times 8370422445201343$
- $\texttt{PDT}(10) = 2^{-72} \times 476288472308334$
- $\texttt{PDT}(11) = 2^{-72} \times 20042553305308$
- $\texttt{PDT}(12) = 2^{-72} \times 623729532807$
- $\texttt{PDT}(13) = 2^{-72} \times 14354889437$
- $\texttt{PDT}(14) = 2^{-72} \times 244322621$
- $\texttt{PDT}(15) = 2^{-72} \times 3075302$
- $\texttt{PDT}(16) = 2^{-72} \times 28626$
- $\texttt{PDT}(17) = 2^{-72} \times 197$
- $\texttt{PDT}(18) = 2^{-72} \times 1$

Our experiment showed that for any $a \geq 509$, $R_a(\mathsf{SampleCDT}_{w=19,\theta=72}, D_{\mathbb{Z}^+,\sigma_{\max}}) \leq 1 + 2^{-80} \leq 1 + \frac{1}{4Q_{\text{bs}}}$, which validates condition (2) for Falcon implementation.

6 Analysis of resistance against timing attacks

In this section, we show that Algorithm 1 is impervious against timing attacks. We formally prove that it is isochronous with respect to σ, μ and the output z (in the sense of Definition 4). We first prove a technical lemma which shows that the number of iterations in the **while** loop of Algorithm 1 is (almost) independent of σ, μ, z.

Lemma 7. *Let $\epsilon \in (0, 1)$, $\mu \in [0, 1]$ and let $\sigma_{\min}, \sigma, \sigma_0$ be standard deviations such that $\eta_{\epsilon}^{+}(\mathbb{Z}^n) = \sigma_{\min} \leq \sigma \leq \sigma_0$. Let $p = \frac{\sigma_{\min} \cdot \sqrt{2\pi}}{2 \cdot \rho_{\sigma_{\max}}(\mathbb{Z}^+)}$. The number of iterations of the* **while** *loop in $\mathsf{SamplerZ}(\sigma, \mu)$ follows a geometric law of parameter*

$$P_{\textbf{true}}(\sigma, \mu) \in p \cdot \left[1, 1 + \frac{(1 + 2^{-80})\epsilon}{n}\right].$$

[3] We are thankful to Thomas Pornin for bringing up this fact.

The proof of Lemma 7 can be found in Appendix B.

Next, we show that Algorithm 1 is perfectly isochronous with respect to z and statistically isochronous (for the Rényi divergence) with respect to σ, μ.

Theorem 8. *Let $\epsilon \in (0,1)$, $\mu \in \mathbb{R}$, let $\sigma_{\min}, \sigma, \sigma_0$ be standard deviations such that $\eta_\epsilon^+(\mathbb{Z}^n) = \sigma_{\min} \leq \sigma \leq \sigma_0$, and let $p = \frac{\sigma_{\min}\cdot\sqrt{2\pi}}{2\cdot\rho_{\sigma_{\max}}(\mathbb{Z}^+)}$ be a constant in $(0,1)$. Suppose that the elementary operations $\{+,-,\times,/\}$ over integer and floating-point numbers are isochronous. The running time of Algorithm 1 follows a distribution $T_{\sigma,\mu}$ such that:*

$$R_a(T_{\sigma,\mu}\|T) \lesssim 1 + \frac{a\epsilon^2 \max(1, \frac{1-p}{p})^2}{n^2(1-p)} = 1 + O\left(\frac{a\epsilon^2}{n^2}\right)$$

for some distribution T independent of its inputs σ, μ and its output z.

Finally, we leverage Theorem 8 to prove that the running time of $\mathsf{SamplerZ}(\sigma, \mu)$ does not help an adversary to break a cryptographic scheme. We consider that the adversary has access to some function $g(\mathsf{SamplerZ}(\sigma, \mu))$ as well as the running time of $\mathsf{SamplerZ}(\sigma, \mu)$: this is intended to capture the fact that in practice the output of $\mathsf{SamplerZ}(\sigma, \mu)$ is not given directly to the adversary, but processed by some function before. For example, in the signature scheme Falcon, samples are processed by algorithms depending on the signer's private key. On the other hand, we consider that the adversary has powerful timing attack capabilities by allowing him to learn the exact runtime of each call to $\mathsf{SamplerZ}(\sigma, \mu)$.

Corollary 9. *Consider an adversary $\mathcal{A}$ making Q_s queries to $g(\mathsf{SamplerZ}(\sigma, \mu))$ for some randomized function g, and solving a search problem with success probability $2^{-\lambda}$ for some $\lambda \geq 1$. With the notations of Theorem 8, suppose that $\max(1, \frac{1-p}{p})^2 \leq n(1-p)$ and $\epsilon \leq \frac{1}{\sqrt{\lambda Q_s}}$. Learning the running time of each call to $\mathsf{SamplerZ}(\sigma, \mu)$ does not increase the success probability of $\mathcal{A}$ by more than a constant factor.*

The proof of Corollary 9 can be found in Appendix D. A nice thing about Corollary 9 is that the conditions required to make it effective are *already met in practice* since they are also required for black-box security of cryptographic schemes. For example, it is systematic to set $\sigma \geq \eta_\epsilon^+(\mathbb{Z}^n)$.

Impact of the scaling factor. The scaling factor $\frac{\sigma_{\min}}{\sigma} \leq \frac{\sigma_{\min}}{\sigma_{\max}}$ is crucial in making our sampler isochronous, as it decorrelates the running time $T_{\sigma,\mu}$ from σ. However, it also impacts the $T_{\sigma,\mu}$, as one can easily show that $T_{\sigma,\mu}$ is proportional to the scaling factor. It is therefore desirable to make it as small as possible. The maximal value of the scaling factor is actually dependent on the cryptographic scheme in which our sampler is used. In Appendix E, we show that for the case of the signature scheme Falcon, $\frac{\sigma_{\min}}{\sigma_{\max}} \leq 1.17^{-2} \approx 0.73$ and the impact of the scaling factor is limited. Moreover, one can easily show that for Peikert's sampler [53], the scaling factor is equal to 1 and has no impact.

7 "Err on the side of Gaussian"

This section focuses on ensuring correct and verified implementations of our proposed isochronous Gaussian sampler. The motivation for this section is to minimize implementation bugs, such as implementation issues with Falcon [56,52] or the famous Heartbleed (CVE-2014-0160) or ROCA vulnerabilities [49] (CVE-2017-15361). We propose a test suite named SAGA (Statistically Acceptable GAussians) in order to verify correct univariate or multivariate Gaussian variables. At the very least, SAGA can act as a "sanity check" for implementers and practitioners. Furthermore,

SAGA is designed to run in a generic fashion, agnostic to the technique used, by only requiring as input a list of univariate (i.e., outputs of SamplerZ) or multivariate (i.e. a set of signatures) Gaussian samples. Although we evaluate SAGA by applying it to Falcon, SAGA is applicable to any lattice-based cryptographic scheme requiring Gaussian sampling, such as other GPV-based signatures [5,13], FrodoKEM [48], identity-based encryption [20,10], and in fully homomorphic encryption [59].

7.1 Univariate tests

The statistical tests we implement here are inspiried by a previous test suite proposal called GLITCH [33]. We use standard statistical tools to validate a Gaussian sampler is operating with the correct mean, standard deviation, skewness, and kurtosis, and finally we check whether it passes a chi-square normality test. Skewness and kurtosis are descriptors of a normal distribution that respectively measure the symmetry and peakedness of a distribution. To view the full statistical analysis of these tests we created a Python class, `UnivariateSamples`, which take as initialization arguments the expected mean (`mu`), expected standard deviation (`sigma`), and the list of observed univariate Gaussian samples (`data`). An example of how this works, as well as its output, is shown in Appendix F.1.

7.2 Multivariate tests

This section details multivariate normality tests. The motivation for these tests is to detect situations where the base Gaussian sampler over the integers is correctly implemented, yet the high-level scheme (e.g. a signature scheme) uses it incorrectly way and ends up with a defective multivariate Gaussian.

Multivariate normality. There are a number of statistical tests which evaluate the normality of multivariate distributions. We found that multivariate normality tests predominantely used in other fields [43,32,14] suffer with size and scaling issues. That is, the large sample sizes we expect to use and the poor power properties of these tests will make a type II error highly likely[4]. In fact, we implemented the Mardia [43] and Henze-Zirkler [32] tests and found, although they worked for small sample sizes, they diverged to produce false negatives for sample sizes ≥ 50 even in small dimensions $n = 64$.

However, the Doornik-Hansen test [16] minimises these issues by using transformed versions of the skewness and kurtosis of the multivariate data, increasing the test's power. We also note that it is much faster (essentially linear in the sample size) than [43,32] (essentially quadratic in the sample size). As with the univariate tests, we created a Python class, denoted `MultivariateSamples`, which outputs four results; two based on the covariance matrix, and two based on the data's normality. An example of how this works, as well as its output, is shown in Appendix F.2.

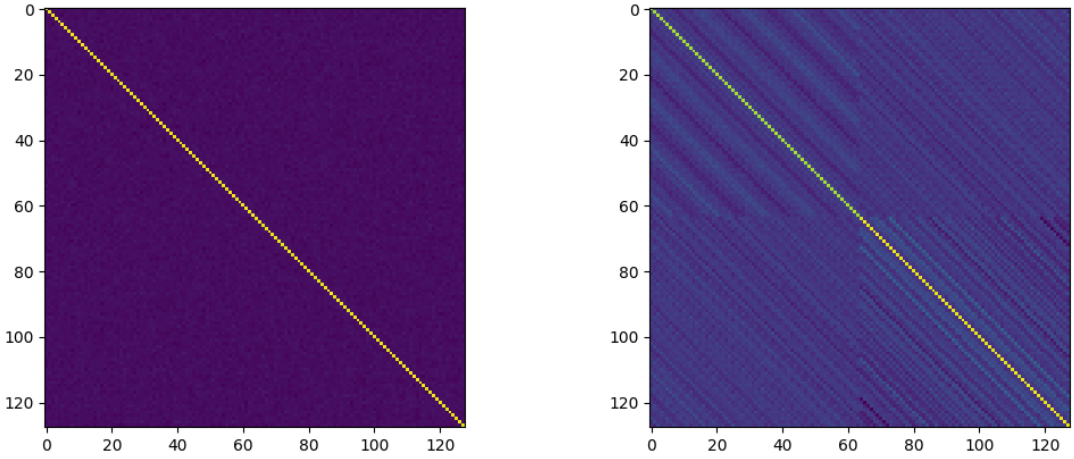
A glitch in the (covariance) matrix. Our second multivariate test asks the following question: *how could someone implement correctly the base sampler, yet subsequently fail to use it properly?* There is no universal answer to that, and one usually has to rely on context, experience and common sense to establish the most likely way this could happen.

For example, in Falcon, univariate samples are linearly combined according to node values of a balanced binary tree computed at key generation (the Falcon tree). If there is an implementation

[4] Type I and type II errors are, respectively, rejection of a true null hypothesis and the non-rejection of a false null hypothesis.

mistake in the procedure computing the tree (during key generation) or when combining the samples (during signing), this effectively results in nodes of the Falcon tree being incorrect or omitted. Such mistakes have a very recognizable effect on the empiric covariance matrix of Falcon signatures: they make them look like block Toeplitz matrices (Figure 1b) instead of (scaled) identity matrices in the nominal case (Figure 1a).

We devised a test which discriminates block Toeplitz covariance matrices against the ones expected from spherical Gaussians. The key idea is rather simple: when adding $O(n)$ coefficients over a (block-)subdiagonal of the empiric covariance matrix, the absolute value of the sum will grow in $O(\sqrt{n})$ if the empiric covariance matrix converges to a scaled identity matrix, and in $O(n)$ if it is block Toeplitz. We use this difference in growth to detect defective Gaussians. While we do not provide a formal proof of our test, in practice it detects reasonably well Gaussians induced by defective Falcon trees. We see proving our test and providing analogues for other GPV-based schemes as interesting questions.



(a) Nominal case

(b) Defective Gaussian

Figure 1: Empiric covariance matrices of Falcon signatures. Figure 1a corresponds to a correct implementation of Falcon. Figure 1b corresponds to an implementation where there is a mistake when constructing the Falcon tree.

Supplementary tests. In the case where normality has been rejected, SAGA also provides a number of extra tests to aid in finding the issues. More details for this can be found in Appendix F.3.

8 Application and Limitations

Our sampler has been implemented by Pornin as part of the new isochronous implementation of Falcon [56]. This implementation can use floating-point hardware or AVX2 instructions when available, but also includes floating-point emulation code that uses only usual integer operations. On ARM Cortex M4 CPUs, which can only support single-precision floating-point instructions,

Table 2: Number of samples per second at 2.5 GHz for our sampler and [64].

Algorithm	Number of samples
This work[5]	1.84×10^6/sec
This work (AVX2)[6]	7.74×10^6/sec
[64] (AVX2)[7]	5.43×10^6/sec

this implementation provides assembly implementations for the core double-precision floating-point operations more than twice faster than the generic emulation. As a result, our sampler can be efficiently implemented on embedded platforms as limited as Cortex M4 CPUs, while some other samplers (e.g. [35] due to a huge code size) are not compact enough to fit embedded platforms.

We perform benchmarks of this sampler implementation on a single Intel Core i7-6500U CPU core clocked at 2.5 GHz. In Table 2 we present the running times of our isochronous sampler. To compare with [64], we scale the numbers to be based on 2.5GHz. Note that for our sampler the number of samples per second is on average for $1.2915 < \sigma \leq 1.8502$ while for [64] $\sigma = 2$ is fixed.

In Table 3 we present the running times of the Falcon isochronous implementation [56] that contains our sampler and compare it with a second non-isochronous implementation nearly identical excepting the base sampler which is a faster lazy CDT sampler, and the rejection sampling which is not scaled by a constant. Compared to the non-isochronous implementation, the isochronous one is about 22% slower, but remains very competitive speed-wise.

Table 3: Falcon signature generation time at 2.5 GHz.

Degree	Non-isochronous (using AVX2)	isochronous (using AVX2)
512	210.88 μs (153.64 μs)	257.33 μs (180.04 μs)
1024	418.76 μs (311.33 μs)	515.28 μs (361.39 μs)

Cache-timing protection. Following this implementation of the proposed sampler also ensures cache-timing protection [25], as the design *should*[8] bypass conditional branches by using a consistant access pattern (using linear searching of the table) and have isochronous runtime. This has been shown to be sufficient in implementations of Gaussian samplers in Frodo [7,48].

Adapting to other schemes. A natural question is how our algorithms could be adapted for other schemes than Falcon, for example [45,20,29,5,12]. An obvious bottleneck seems to be the size of the CDT used in SampleCDT, which is linear in the standard deviation. For larger standard deviations, where linear searching becomes impractical, convolutions can be used to reduce σ, and thus the runtime of the search algorithm [55,37]. It would also be interesting to see if the DDG tree-based method of [35] has better scalability than our CDT-based method, in which case we would recommend it for larger standard deviations. On the other hand, once the base sampler is implemented, we do not see any obvious obstacle for implementing our whole framework. For example, [12] or using Peikert's sampler [53] (in Falcon) entail a small constant

[5] [56] standard double-precision floating-point (IEEE 754) with SHAKE256.
[6] [56] AVX2 implementation wth eight ChaCha20 instances in parallel (AVX2).
[7] [64] constant-time implementation with hardware AES256 (AES-NI).
[8] Compilers may alter the design, thus one should always verify the design post-compilation.

number of standard deviations, therefore the rejection step would be very efficient once a base sampler for each standard deviation is implemented.

Advantages and limitations. Our sampler has an acceptance rate $\approx \frac{\sigma_{\min}}{\sigma_{\max}+0.4}$ making it especially suitable when $\sigma_{\min}$ and $\sigma_{\max}$ are close. In particular, our sampler is, so far, the fastest isochronous sampler for the parameters in Falcon. However, the larger the gap between $\sigma_{\min}$ and $\sigma_{\max}$, the lower the acceptance rate. In addition, our sampler uses a cummulative distribution table (CDT) which is accessed in an isochronous way. This table grows when $\sigma_{\max}$ grows, while making both running time and memory usage larger. When $\sigma_{\max}$ is large or far from $\sigma_{\min}$, there exist faster isochronous samplers based on convolution [47] and rejection sampling [64][9] techniques.

Acknowledgements

We thank Léo Ducas for helpful suggestions. We also thank Thomas Pornin and Mehdi Tibouchi for useful discussions. The first and second authors were supported by the project PQ Cybersecurity (Innovate UK research grant 104423). The third and fourth authors were supported by BPI-France in the context of the national project RISQ (P141580), and by the European Union PROMETHEUS project (Horizon 2020 Research and Innovation Program, grant 780701). The fourth author was also supported by ANRT under the program CIFRE N2016/1583.

[9] The constant-time sampler in [64] may still reveal σ.

References

1. Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient lattice (H)IBE in the standard model. In Gilbert [31], pages 553–572.
2. Joachim Ahrens and Ulrich Dieter. Extension of forsythe's method for random sampling from the normal distribution. *Mathematics of computation*, 27:927–937, 1973.
3. Shi Bai, Adeline Langlois, Tancrède Lepoint, Damien Stehlé, and Ron Steinfeld. Improved security proofs in lattice-based cryptography: Using the Rényi divergence rather than the statistical distance. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part I*, volume 9452 of *LNCS*, pages 3–24. Springer, Heidelberg, November / December 2015.
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. Cryptology ePrint Archive, Report 2019/511, 2019.
5. Pauline Bert, Pierre-Alain Fouque, Adeline Roux-Langlois, and Mohamed Sabt. Practical implementation of ring-SIS/LWE based signature and IBE. In Tanja Lange and Rainer Steinwandt, editors, *Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018*, pages 271–291. Springer, Heidelberg, 2018.
6. Nina Bindel, Sedat Akleylek, Erdem Alkim, Paulo S. L. M. Barreto, Johannes Buchmann, Edward Eaton, Gus Gutoski, Juliane Kramer, Patrick Longa, Harun Polat, Jefferson E. Ricardini, and Gustavo Zanon. qTESLA. Technical report, National Institute of Standards and Technology, 2019. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions.
7. Joppe W. Bos, Craig Costello, Léo Ducas, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Ananth Raghunathan, and Douglas Stebila. Frodo: Take off the ring! Practical, quantum-secure key exchange from LWE. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 1006–1018. ACM Press, October 2016.
8. Joachim Breitner and Nadia Heninger. Biased nonce sense: Lattice attacks against weak ecdsa signatures in cryptocurrencies. In Ian Goldberg and Tyler Moore, editors, *Financial Cryptography and Data Security*, pages 3–20, Cham, 2019. Springer International Publishing.
9. Leon Groot Bruinderink, Andreas Hülsing, Tanja Lange, and Yuval Yarom. Flush, gauss, and reload - A cache attack on the BLISS lattice-based signature scheme. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *CHES 2016*, volume 9813 of *LNCS*, pages 323–345. Springer, Heidelberg, August 2016.
10. Peter Campbell and Michael Groves. Practical post-quantum hierarchical identity-based encryption. 16th IMA International Conference on Cryptography and Coding, 2017.
11. David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. Bonsai trees, or how to delegate a lattice basis. In Gilbert [31], pages 523–552.
12. Yilei Chen, Nicholas Genise, and Pratyay Mukherjee. Approximate trapdoors for lattices and smaller hash-and-sign signatures. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019*, volume 11923 of *LNCS*, pages 3–32. Springer, 2019.
13. Yilei Chen, Nicholas Genise, and Pratyay Mukherjee. Approximate trapdoors for lattices and smaller hash-and-sign signatures. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part III*, volume 11923 of *LNCS*, pages 3–32. Springer, Heidelberg, December 2019.
14. David Roxbee Cox and NJH Small. Testing multivariate normality. *Biometrika*, 65(2):263–272, 1978.
15. Ralph B D'Agostino, Albert Belanger, and Ralph B D'Agostino Jr. A suggestion for using powerful and informative tests of normality. *The American Statistician*, 44(4):316–321, 1990.
16. Jurgen A Doornik and Henrik Hansen. An omnibus test for univariate and multivariate normality. *Oxford Bulletin of Economics and Statistics*, 70:927–939, 2008.
17. Yusong Du, Baodian Wei, and Huang Zhang. A rejection sampling algorithm for off-centered discrete gaussian distributions over the integers. *Science China Information Sciences*, 62(3):39103, Sep 2018.
18. Léo Ducas. *Signatures fondées sur les réseaux euclidiens : attaques, analyses et optimisations*. Theses, École Normale Supérieure, 2013.
19. Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. Lattice signatures and bimodal Gaussians. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 40–56. Springer, Heidelberg, August 2013.
20. Léo Ducas, Vadim Lyubashevsky, and Thomas Prest. Efficient identity-based encryption over NTRU lattices. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part II*, volume 8874 of *LNCS*, pages 22–41. Springer, Heidelberg, December 2014.
21. Léo Ducas and Phong Q. Nguyen. Faster Gaussian lattice sampling using lazy floating-point arithmetic. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 415–432. Springer, Heidelberg, December 2012.
22. Nagarjun C Dwarakanath and Steven D Galbraith. Sampling from discrete gaussians for lattice-based cryptography on a constrained device. *Applicable Algebra in Engineering, Communication and Computing*, 25(3):159–180, 2014.

23. Thomas Espitau, Pierre-Alain Fouque, Benoît Gérard, and Mehdi Tibouchi. Side-channel attacks on BLISS lattice-based signatures: Exploiting branch tracing against strongSwan and electromagnetic emanations in microcontrollers. In Thuraisingham et al. [60], pages 1857–1874.
24. Gerald Estrin. Organization of computer systems: The fixed plus variable structure computer. In *Papers Presented at the May 3-5, 1960, Western Joint IRE-AIEE-ACM Computer Conference*, IRE-AIEE-ACM '60 (Western), pages 33–40, New York, NY, USA, 1960. ACM.
25. Adrien Facon, Sylvain Guilley, Matthieu Lec'Hvien, Alexander Schaub, and Youssef Souissi. Detecting cache-timing vulnerabilities in post-quantum cryptography algorithms. In *2018 IEEE 3rd International Verification and Security Workshop (IVSW)*, pages 7–12. IEEE, 2018.
26. James J. Filliben and Alan Heckert. *1.3.3.24. Quantile-Quantile Plot*. 2013.
27. George E. Forsythe. Von neumann's comparison method for random sampling from the normal and other distributions. *Mathematics of Computation*, 26(120):817–826, 1972.
28. Nicolas Gama, Nick Howgrave-Graham, and Phong Q. Nguyen. Symplectic lattice reduction and NTRU. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 233–253. Springer, Heidelberg, May / June 2006.
29. Nicholas Genise and Daniele Micciancio. Faster Gaussian sampling for trapdoor lattices with arbitrary modulus. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 174–203. Springer, Heidelberg, April / May 2018.
30. Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 197–206. ACM Press, May 2008.
31. Henri Gilbert, editor. *EUROCRYPT 2010*, volume 6110 of *LNCS*. Springer, Heidelberg, May / June 2010.
32. N Henze and B Zirkler. A class of invariant consistent tests for multivariate normality. *Communications in Statistics-Theory and Methods*, 19(10):3595–3617, 1990.
33. James Howe and Máire O'Neill. GLITCH: A discrete gaussian testing suite for lattice-based cryptography. In *Proceedings of the 14th International Joint Conference on e-Business and Telecommunications (ICETE 2017) - Volume 4: SECRYPT, Madrid, Spain, July 24-26, 2017.*, pages 413–419, 2017.
34. Andreas Hülsing, Tanja Lange, and Kit Smeets. Rounded Gaussians - fast and secure constant-time sampling for lattice-based crypto. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part II*, volume 10770 of *LNCS*, pages 728–757. Springer, Heidelberg, March 2018.
35. Angshuman Karmakar, Sujoy Sinha Roy, Frederik Vercauteren, and Ingrid Verbauwhede. Pushing the speed limit of constant-time discrete Gaussian sampling. A case study on the Falcon signature scheme. In *Proceedings of the 56th Annual Design Automation Conference 2019*, pages 1–6, 2019.
36. Charles F. F. Karney. Sampling exactly from the normal distribution. *ACM Trans. Math. Softw.*, 42(1):3:1–3:14, January 2016.
37. Ayesha Khalid, James Howe, Ciara Rafferty, Francesco Regazzoni, and Máire O'Neill. Compact, scalable, and efficient discrete gaussian samplers for lattice-based cryptography. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2018.
38. Selcuk Korkmaz, Dincer Goksuluk, and Gokmen Zararsiz. Mvn: An r package for assessing multivariate normality. *The R Journal*, 6(2):151–162, 2014.
39. Vadim Lyubashevsky. Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 598–616. Springer, Heidelberg, December 2009.
40. Vadim Lyubashevsky, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-DILITHIUM. Technical report, National Institute of Standards and Technology, 2019. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions.
41. Vadim Lyubashevsky and Daniel Wichs. Simple lattice trapdoor sampling from a broad class of distributions. In Jonathan Katz, editor, *PKC 2015*, volume 9020 of *LNCS*, pages 716–730. Springer, Heidelberg, March / April 2015.
42. Prasanta Chandra Mahalanobis. On the generalized distance in statistics. National Institute of Science of India, 1936.
43. Kanti V Mardia. Measures of multivariate skewness and kurtosis with applications. *Biometrika*, 57(3):519–530, 1970.
44. Carlos Aguilar Melchor and Thomas Ricosset. CDT-Based Gaussian Sampling: From Multi to Double Precision. *IEEE Trans. Computers*, 67(11):1610–1621, 2018.
45. Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 700–718. Springer, Heidelberg, April 2012.
46. Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on gaussian measures. *SIAM J. Comput.*, 37(1):267–302, 2007.

47. Daniele Micciancio and Michael Walter. Gaussian sampling over the integers: Efficient, generic, constant-time. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 455–485. Springer, Heidelberg, August 2017.
48. Michael Naehrig, Erdem Alkim, Joppe Bos, Léo Ducas, Karen Easterbrook, Brian LaMacchia, Patrick Longa, Ilya Mironov, Valeria Nikolaenko, Christopher Peikert, Ananth Raghunathan, and Douglas Stebila. FrodoKEM. Technical report, National Institute of Standards and Technology, 2019. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions.
49. Matús Nemec, Marek Sýs, Petr Svenda, Dusan Klinec, and Vashek Matyas. The return of coppersmith's attack: Practical factorization of widely used RSA moduli. In Thuraisingham et al. [60], pages 1631–1648.
50. NIST et al. OFFICIAL COMMENT: CRYSTALS-DILITHIUM, 2018. https://groups.google.com/a/list.nist.gov/d/msg/pqc-forum/aWxC2ynJDLE/YOsMJ2ewAAAJ.
51. NIST et al. Footguns as an axis for security analysis, 2019. https://groups.google.com/a/list.nist.gov/forum/#!topic/pqc-forum/l2iYk-8sGnI last accessed 23-09-2019.
52. NIST et al. OFFICIAL COMMENT: Falcon (bug & fixes), 2019. https://groups.google.com/a/list.nist.gov/forum/#!topic/pqc-forum/7Z8x5AMXy8s last accessed on 23-09-2019.
53. Chris Peikert. An efficient and parallel Gaussian sampler for lattices. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 80–97. Springer, Heidelberg, August 2010.
54. Peter Pessl, Leon Groot Bruinderink, and Yuval Yarom. To BLISS-B or not to be: Attacking strongSwan's implementation of post-quantum signatures. In Thuraisingham et al. [60], pages 1843–1855.
55. Thomas Pöppelmann, Léo Ducas, and Tim Güneysu. Enhanced lattice-based signatures on reconfigurable hardware. In Lejla Batina and Matthew Robshaw, editors, *CHES 2014*, volume 8731 of *LNCS*, pages 353–370. Springer, Heidelberg, September 2014.
56. Thomas Pornin. New Efficient, Constant-Time Implementations of Falcon. Cryptology ePrint Archive, Report 2019/893, 2019.
57. Thomas Prest. Sharper bounds in lattice-based cryptography using the Rényi divergence. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 347–374. Springer, Heidelberg, December 2017.
58. Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. FALCON. Technical report, National Institute of Standards and Technology, 2019. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions.
59. Microsoft SEAL (release 3.4). https://github.com/Microsoft/SEAL, October 2019. Microsoft Research, Redmond, WA.
60. Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors. *ACM CCS 2017*. ACM Press, October / November 2017.
61. Mehdi Tibouchi and Alexandre Wallet. One Bit is All It Takes: A Devastating Timing Attack on BLISS's Non-Constant Time Sign Flips. MathCrypt 2019, 2019.
62. John von Neumann. Various techniques used in connection with random digits. *National Bureau of Standards, Applied Math Series*, 12:36–38, 11 1950.
63. Michael Walter. Sampling the integers with low relative error. In Johannes Buchmann, Abderrahmane Nitaj, and Tajje-eddine Rachidi, editors, *AFRICACRYPT 2019*, volume 11627 of *LNCS*, pages 157–180. Springer, 2019.
64. Raymond K. Zhao, Ron Steinfeld, and Amin Sakzad. Compact and scalable arbitrary-centered discrete gaussian sampling over integers. Cryptology ePrint Archive, Report 2019/1011, 2019.
65. Raymond K Zhao, Ron Steinfeld, and Amin Sakzad. Facct: Fast, compact, and constant-time discrete gaussian sampler over integers. *IEEE Transactions on Computers*, 2019.

A Proof of Theorem 6

To estimate the security loss with the replacement of $D_{\mathbb{Z},\sigma,\mu}$ by SamplerZ, we introduce an intermediate case where ApproxExp outputs a perfect value. The 3 cases are defined as follows.

1. (Ideal) The ideal $D_{\mathbb{Z},\sigma,\mu}$ is called. By Proposition 5, it is the same as considering ApproxExp $=$ exp and BaseSampler $= D_{\mathbb{Z}^+,\sigma_{\max}}$.
2. (Inter) Only the exponential is considered as perfect, i.e. ApproxExp $=$ exp is assumed.
3. (Real) The imperfect SamplerZ is called.

We recall that λ_{Real} (resp. λ_{Ideal}) is the security parameter of the Real (resp. Ideal) case. We aim at computing $\Delta\lambda = \lambda_{\text{Ideal}} - \lambda_{\text{Real}}$.

We denote by $a \coloneqq 2 \cdot \lambda_{\text{REAL}} + 1$. The values $R_a(\text{REAL}, \text{INTER})$ and $R_a(\text{INTER}, \text{IDEAL})$ will be used to quantify the distance between each case. Let E be an event breaking the scheme. Let δ_{IDEAL} (resp. δ_{INTER}, δ_{REAL}) be the probability that this event occurs in the use of the IDEAL (resp. INTER, REAL) case. We consider that the number of queries to the BaseSampler (resp. ApproxExp) is bounded by Q_{bs} (resp. Q_{exp}). By data processing and probability preservation of the Rényi divergence:

$$\delta_{\text{IDEAL}} \geq \delta_{\text{INTER}}^{\frac{a}{a-1}} / R_a(\text{INTER}^{Q_{\text{bs}}}, \text{IDEAL}^{Q_{\text{bs}}}) \geq \delta_{\text{INTER}}^{\frac{a}{a-1}} / R_a(\text{INTER}, \text{IDEAL})^{Q_{\text{bs}}}$$

$$\delta_{\text{INTER}} \geq \delta_{\text{REAL}}^{\frac{a}{a-1}} / R_a(\text{REAL}^{Q_{\text{exp}}}, \text{INTER}^{Q_{\text{exp}}}) \geq \delta_{\text{REAL}}^{\frac{a}{a-1}} / R_a(\text{REAL}, \text{INTER})^{Q_{\text{exp}}}.$$

By definition, $\delta_{\text{REAL}} \geq 2^{-\lambda_{\text{REAL}}}$, thus, the second equation can be upper bounded using $\delta_{\text{REAL}}^{\frac{a}{a-1}} \geq \delta_{\text{REAL}}/\sqrt{2}$. By combining it,

$$\delta_{\text{INTER}} \geq \delta_{\text{REAL}} / \left(\sqrt{2} \cdot R_a(\text{REAL}, \text{INTER})^{Q_{\text{exp}}}\right).$$

And thus,

$$\begin{aligned}\delta_{\text{IDEAL}} &\geq \delta_{\text{REAL}}^{\frac{a}{a-1}} \cdot \left(\sqrt{2}^{\frac{a}{a-1}} \cdot R_a(\text{REAL}, \text{INTER})^{\frac{aQ_{\text{exp}}}{a-1}} R_a(\text{INTER}, \text{IDEAL})^{Q_{\text{bs}}}\right)^{-1} \\ &\geq \delta_{\text{REAL}} \cdot \left(\sqrt{2}^{\frac{a}{a-1}+1} \cdot R_a(\text{REAL}, \text{INTER})^{\frac{aQ_{\text{exp}}}{a-1}} R_a(\text{INTER}, \text{IDEAL})^{Q_{\text{bs}}}\right)^{-1}\end{aligned}$$

So,

$$\Delta\lambda = \log_2 \left(\sqrt{2}^{\frac{a}{a-1}+1} \cdot R_a(\text{REAL}, \text{INTER})^{\frac{aQ_{\text{exp}}}{a-1}} \cdot R_a(\text{INTER}, \text{IDEAL})^{Q_{\text{bs}}}\right)$$

Let us now use the conditions to get a concrete upper bound on $\Delta\lambda$. First, suppose that condition (1) is verified. We use $a \coloneqq 2 \cdot \lambda_{\text{REAL}} + 1$, then, for all $x < 0$:

$$1 - \sqrt{\frac{a-1}{2 \cdot a^2 \cdot Q_{\text{exp}}}} \leq \frac{P(x)}{\mathsf{ApproxExp}(x)} \leq 1 + \sqrt{\frac{a-1}{2 \cdot a^2 \cdot Q_{\text{exp}}}}.$$

An application of Lemma 3 yields to $R_a(\text{REAL}, \text{INTER}) \leq 1 + \frac{a-1}{4aQ_{\text{exp}}}$.

Secondly, suppose that condition (2) is verified. Recall that $BG_{\sigma_{\max}}$ denotes the ideal distribution of z before rejection sampling (Step 4). Let us denote by $\overline{BG_{\sigma_{\max}}}$ the distribution of z before the rejection sampling when BaseSampler is not perfect. The next step s in SamplerZ algorithm consist in multiplying the output distribution by $z \mapsto \frac{\sigma_{\min}}{\sigma} \exp\left(\frac{(z-\mu)^2}{2\sigma^2} - \frac{z_0^2}{2\sigma_{\max}^2}\right)$. By data processing, we get

$$R_a(\text{INTER}, \text{IDEAL}) \leq R_a(\overline{BG_{\sigma_{\max}}}, BG_{\sigma_{\max}})$$

Then, since (considering the distribution of b as perfectly uniform)

$$R_a(\text{INTER}, \text{IDEAL}) \leq R_a\left((2b-1)\mathsf{BaseSampler} + b, (2b-1)D_{\mathbb{Z}^+, \sigma_{\max}} + b\right),$$

we re-apply data processing and obtain

$$\begin{aligned}R_a(\text{INTER}, \text{IDEAL}) &\leq R_a(\mathsf{BaseSampler}, D_{\mathbb{Z}^+, \sigma_{\max}}) \\ &\leq 1 + \tfrac{1}{4 \cdot Q_{\text{bs}}}.\end{aligned}$$

Wrapping up,

$$\begin{aligned}\Delta\lambda &= \log_2\left(\sqrt{2}^{\frac{a}{a-1}+1}\cdot R_a(\textsc{Real},\textsc{Inter})^{\frac{aQ_{\exp}}{a-1}}\cdot R_a(\textsc{Inter},\textsc{Ideal})^{Q_{\mathrm{bs}}}\right)\\ &\leq \log_2\left(\sqrt{2}^{\frac{a}{a-1}+1}\cdot\left(1+\tfrac{a-1}{4aQ_{\exp}}\right)^{\frac{aQ_{\exp}}{a-1}}\cdot\left(1+\tfrac{1}{4\cdot Q_{\mathrm{bs}}}\right)^{Q_{\mathrm{bs}}}\right).\end{aligned}$$

Using the inequality $\left(1+\frac{x}{n}\right)^n \leq \exp(x)$ for $x, n > 0$,

$$\begin{aligned}\Delta\lambda &\leq \log_2\left(\sqrt{2}^{\frac{a}{a-1}+1}\cdot\exp(1/4)^2\right)\\ &= \log_2\left(\sqrt{2}^{\frac{a}{a-1}+1}\cdot 2\right)\\ &\leq 2.\end{aligned}$$

B Proof of Lemma 7

Proof. We note $p_1(z)$ the probability that $z \in \mathbb{Z}$ (with a uniquely associated z_0) is output by SamplerZ in a given iteration. It holds that:

$$p_1(z) = \underbrace{\frac{\rho_{\sigma_{\max}}(z_0)}{\rho_{\sigma_{\max}}(\mathbb{Z}^+)}}_{\mathbb{P}[\mathsf{BaseSampler}\to z_0]}\cdot\underbrace{\frac{1}{2}}_{\mathbb{P}[b]}\cdot\underbrace{\frac{\sigma_{\min}}{\sigma}\cdot\frac{\rho_{\sigma,\mu}(z)}{\rho_{\sigma_{\max}}(z_0)}}_{\mathbb{P}[\mathsf{AcceptSample}\to\mathbf{true}|z]} = \frac{\sigma_{\min}\cdot\rho_{\sigma,\mu}(z)}{2\cdot\sigma\cdot\rho_{\sigma_{\max}}(\mathbb{Z}^+)}.$$

The probability $P_{\mathbf{true}}$ that ($\mathsf{AcceptSample}\to\mathbf{true}$) in a given iteration is:

$$P_{\mathbf{true}} = \mathbb{P}[\mathsf{AcceptSample}\to\mathbf{true}] = \sum_z p_1(z) = \frac{\sigma_{\min}\cdot\rho_{\sigma,\mu}(\mathbb{Z})}{2\cdot\sigma\cdot\rho_{\sigma_{\max}}(\mathbb{Z}^+)}. \tag{5}$$

One can see in (5) that $P_{\mathbf{true}}$ is independent of the output z. This is unsurprising since a different z is picked at each new iteration of the **while** loop, and each iteration's running time is constant.

However, it is not obvious from (5) that $P_{\mathbf{true}}$ is independent of σ and μ; we now show that it is essentially the case. Since $\sigma \geq \eta^+_\epsilon(\mathbb{Z}^n) \geq \eta^+_{\epsilon/n}(\mathbb{Z}) \geq \eta_{\epsilon/n}(\mathbb{Z})$, it holds from [30, Lemma 2.7] that:

$$\rho_{\sigma,\mu}(\mathbb{Z}) \in \left[\frac{1-\epsilon/n}{1+\epsilon/n}, 1\right]\cdot\rho_\sigma(\mathbb{Z}). \tag{6}$$

It is now helpful to bound $\rho_\sigma(\mathbb{Z})$. By the Poisson summation formula:

$$\rho_\sigma(\mathbb{Z}) = \sigma\sqrt{2\pi}\cdot\left(1+2\sum_{i\geq 1}\exp(-2i^2\pi^2\sigma^2)\right). \tag{7}$$

For any $\sigma > 1$, it holds that $\sum_{i\geq 1}\exp(-2i^2\pi^2\sigma^2) \in \exp(-2\pi^2\sigma^2)\cdot\left[1, 1+2^{-80}\right]$. Moreover, it follows from (3) that $\exp(-2\pi^2\sigma^2) \leq \frac{\epsilon}{2n}$. Combined this fact with (7) yields:

$$\sigma\sqrt{2\pi} \leq \rho_\sigma(\mathbb{Z}) \leq \sigma\sqrt{2\pi}\cdot\left(1+(1+2^{-80})\cdot\frac{\epsilon}{n}\right) \tag{8}$$

Finally, combining (5), (6) and (8) yields:

$$P_{\mathbf{true}}(\sigma,\mu) \in \frac{\sigma_{\min}\cdot\sqrt{2\pi}}{2\cdot\rho_{\sigma_{\max}}(\mathbb{Z}^+)}\cdot\left(1, 1+\frac{(1+2^{-80})\epsilon}{n}\right). \tag{9}$$

This concludes the proof. □

C Proof of Theorem 8

Before proving Theorem 8, we will need a preliminary lemma. Note that this lemma cannot be proven in a black-box way using Lemma 3 since the relative error between any two distrinct geometric distributions is infinite.

Lemma 10. *Let $\mathcal{P}$ and $\mathcal{Q}$ be geometric distributions of parameters $p, q \geq C$ for a constant $C \geq 0$. Suppose there exists $\delta = o(1/(a+1))$ such that:*

$$\begin{aligned} e^{-\delta} \leq \quad & p/q \quad & \leq e^{\delta}, \\ e^{-\delta} \leq (1-p)/(1-q) & & \leq e^{\delta}. \end{aligned}$$

Then the Rényi divergence between $\mathcal{P}$ and $\mathcal{Q}$ is bounded as follows:

$$R_a(\mathcal{P}\|\mathcal{Q}) \lesssim 1 + \frac{a(1-p)\delta^2}{p^2}\left(\sim 1 + \frac{a(1-q)\delta^2}{q^2}\right).$$

Proof. The beginning of the proof follows the one of [57, Lemma 3], but makes a more precise estimation. Let $f_a : (x, y) \mapsto \frac{y^a}{(x+y)^{a-1}}$. We compute values of f_a and its derivatives around $(0, y)$:

$$\begin{aligned} f_a(x,y) &= y && \text{for } x = 0 \\ \tfrac{\partial f_a}{\partial x}(x,y) &= 1-a && \text{for } x = 0 \\ \tfrac{\partial^2 f_a}{\partial x^2}(x,y) &= a(a-1)y^a(x+y)^{-a-1} && \\ &\leq \tfrac{a(a-1)}{e^{-(a+1)\delta}y} && \text{for } |x+y| \leq e^{\delta}\cdot y \end{aligned}$$

We now use partial Taylor bounds. If $|x_k| \leq (e^{\delta_k} - 1)\cdot y_k$, then:

$$f_a(x_k, y_k) \leq f_a(0, y) + \frac{\partial f_a}{\partial x}(0, y_k)\cdot x_k + \frac{a(a-1)(e^{\delta_k}-1)^2}{2e^{-(a+1)\delta_k}}\cdot y_k$$

We take $y_k = \mathcal{P}(k)$ and $x_k = \mathcal{Q}(k) - \mathcal{P}(k)$, and note that $e^{-k\delta} \leq \mathcal{P}(k)/\mathcal{Q}(k) \leq e^{k\delta}$, hence $\delta_k \leq k\delta$. Summing all over the support of $\mathcal{P}$ gives:

$$\begin{aligned} R_a^a(\mathcal{P}\|\mathcal{Q}) &\leq 1 + \frac{a(a-1)}{2}\sum_{k\in\mathbb{Z}^+}\frac{(e^{k\delta}-1)^2}{e^{-(a+1)k\delta}}\cdot(1-p)^{k-1}p && (10) \\ &\leq 1 + \frac{pa(a-1)}{2(1-p)}\cdot\left(\frac{2(1-p)^2\delta^2}{p^3} + O\left(\frac{a\delta^3(p-1)^3}{p^4}\right)\right) && (11) \\ &\lesssim 1 + \frac{a(a-1)(1-p)\delta^2}{p^2} \end{aligned}$$

To compute the sum in (10), we expanded $(e^{k\delta}-1)^2$, which then gives us three distinct geometric sums which all converge since $\delta = o(1/(a+1))$, and for which closed formulae are known. A tedious but easy Taylor expansion then gives us 11, at which point we can conclude. □

We now prove Theorem 8.

Proof. Let T_0 denote the running time of one iteration of the **while** loop in Algorithm 1. It is clear that steps 2 to 5 are isochronous. On the other hand, it is also clear that AcceptSample (Algorithm 2) is isochronous; indeed, all its atomic operations are isochronous and each iteration of its **while** loop has a constant probability $1 - 2^{-8}$ of being the last one. We can conclude that T_0 follows a distribution which is independent of σ, μ, z.

Let us denote by $I_{\sigma,\mu}$ (resp. I) the number of iterations of the **while** loop when each iteration accepts with probability $P_{\mathbf{true}}$ (resp. p). By Lemma 7, $I_{\sigma,\mu}$ (resp. I) follows a geometric law of parameter $P_{\mathbf{true}}$ (resp. p). The relative error between $P_{\mathbf{true}}$ and p is upper bounded by $\delta = \frac{(1+2^{-80})\epsilon}{n} \cdot \max(1, \frac{p}{1-p})$. It then follows from Lemma 10 that:

$$\begin{aligned} R_a(I_{\sigma,\mu}\|I) &\lesssim 1 + \frac{a(1-p)\delta^2}{p^2} \\ &\lesssim 1 + \frac{a\epsilon^2 \max(1, \frac{1-p}{p})^2}{n^2(1-p)} \end{aligned}$$

Finally, the total running time T of Algorithm 1 is a function of the running time of each iteration and the number of iterations: $T = f(T_0, I)$ for some function f. This allows to apply once again the data-processing inequality:

$$R_a(T_{\sigma,\mu}\|T) = R_a(f(T_0, I_{\sigma,\mu})\|f(T_0, I)) \leq R_a(I_{\sigma,\mu}\|I),$$

which concludes the proof. □

D Proof of Corollary 9

Proof. Let D denote the output distribution of $g(\mathsf{SamplerZ}(\sigma, \mu))$. In the ideal case, we can consider without loss of generality that the adversary can query the joint distribution (D, T), where T is as in the proof of Theorem 8 and is thus independent from σ, μ. In the real case, the adversary learns the runtime of each call to $\mathsf{SamplerZ}(\sigma, \mu)$. Since we showed in the proof of Theorem 8 that the runtime is independent of the output z, we can model it as the distribution $T_{\sigma,\mu}$ described in the proof of Theorem 8, and the adversary has access to $(D, T_{\sigma,\mu})$.

Let P_0, P_1 denote the success probability of $\mathcal{A}$ in the ideal and real cases, respectively. Taking $a = \lambda$ gives:

$$\begin{aligned} P_1 &\leq \left[P_0 \cdot R_a((D, T_{\sigma,\mu})^{Q_s}\|(D, T)^{Q_s})\right]^{(a-1)/a} && (12) \\ &\leq \left[P_0 \cdot R_a((D, T_{\sigma,\mu})\|(D, T))^{Q_s}\right]^{(a-1)/a} && (13) \\ &\leq \left[P_0 \cdot R_a(T_{\sigma,\mu}\|T)^{Q_s}\right]^{(a-1)/a} && (14) \\ &\lesssim \left[P_0 \cdot \left(1 + \frac{a\epsilon^2 \max(1, \frac{1-p}{p})^2}{n^2(1-p)}\right)^{Q_s}\right]^{(a-1)/a} && (15) \\ &\lesssim \left[P_0 \cdot \left(1 + \frac{1}{n \cdot Q_s}\right)^{Q_s}\right]^{(\lambda-1)/\lambda} && (16) \\ &\lesssim 2^{-(\lambda-1)} \cdot e^{1/n} && (17) \end{aligned}$$

Hereabove, (12) uses the probability preservation property of the Renyi divergence, and (13) uses its multiplicativity. We previously showed that D and $T_{\sigma,\mu}$ are independent, thus we can discard D in (14). We then apply Theorem 8 to get (15). We replace ϵ by $\frac{1}{\sqrt{\lambda Q_s}}$, take $a = \lambda$ and $\max(1, \frac{1-p}{p})^2 \leq n(1-p)$ to obtain (16). Finally, we use the identity $(1 + x/k)^k \leq e^x$ and replace P_0 by $2^{-\lambda}$ to get (17). □

E Impact of the scaling factor in Falcon

We now study the impact of the scaling factor $\frac{\sigma_{\min}}{\sigma} \leq \frac{\sigma_{\min}}{\sigma_{\max}}$ on the running time for the particular case of Falcon. There, each σ verifies $\sigma_{\min} \leq \sigma \leq \sigma_{\max}$, where $\sigma_{\min} = \eta_\epsilon^+(\mathbb{Z}^n)$ and $\sigma_{\max} = \sigma_{\min} \cdot \frac{\max_i \|\tilde{\mathbf{b}_i}\|}{\min_i \|\tilde{\mathbf{b}_i}\|}$. The $\tilde{\mathbf{b}_i}$ are the Gram-Schmidt vectors of the secret, short basis $\mathbf{B}$. In Falcon, it holds that:

$$\max_i \|\tilde{\mathbf{b}_i}\| \leq 1.17\sqrt{q} \tag{18}$$

$$\min_i \|\tilde{\mathbf{b}_i}\| \geq \sqrt{q}/1.17 \tag{19}$$

By construction, (18) is true (Falcon enforces it at key generation). To prove (19), we rely on a peculiar property of Falcon's private bases: *symplecticity*. Let f, g, F, G be such that $fG - gF = q$. Let:

$$\mathbf{J} = \left[\begin{array}{c|c} 0 & 1 \\ \hline -1 & 0 \end{array}\right] \text{ and } \mathbf{B} = \left[\begin{array}{c|c} g & -f \\ \hline G & -F \end{array}\right].$$

This form of $\mathbf{B}$ is indeed the one used in Falcon. It has been observed in [28] that $\mathbf{B}$ is q-symplectic, that is, it verifies:

$$\mathbf{B}^t \times \mathbf{J} \times \mathbf{B} = q \cdot \mathbf{J}. \tag{20}$$

As per [28, Corollary 1], this implies that for any i, $\|\tilde{\mathbf{b}}_{2n+1-i}\| = q/\|\tilde{\mathbf{b}}_i\|$. Combining this with (18) yields (19). Thus $\frac{\sigma_{\min}}{\sigma_{\max}} \leq (1.17)^{-2} \approx 0.73$, which means a non-negligible but reasonable impact on the running time of the sampler.

F Additional information on SAGA

F.1 Univariate tests

An example of the standard outputs of the univariate tests in SAGA is shown in Listing 1.1.

```
>>> python3              # initialize python
>>> import saga          # import the test suite
>>> mu = -0.920619       # define the centre
>>> sigma = 1.711864     # define the standard deviation
>>> data = [-1, 2, -4, 0, -2, 1, -2, -3, -1, 1, -1, 0, 0, 1, -4, -1, -2, -2, -1,
    0, 1, -1, 2, -3, 2, 0, -1, -2, 0, -3, -1, -2, -1, 1, -5, -1, -2, -2, -1, 0, 2,
     1, 0, 0, 1, -1, -2, -2, -1, 0, 2, -2, -1, -3, 0, 0, 0, -2, 0, 0, 0, -3, -4,
    0, 1, -1, 0, -1, 1, -3, 0, 0, -3, 0, -4, -1, -2, 0, 0, -2, -2, -1, 1, -1, 0,
    -2, -2, -2, 0, -1, -4, -2, 0, -2, -2, 1, -1, 0, -3, -1]
>>>
>>> res = saga.UnivariateSamples(mu, sigma, data)
>>> res

Testing a Gaussian sampler with center = -0.920619 and sigma = 1.711864
Number of samples: 100

Moments  |   Expected     Empiric
---------+----------------------
Mean:    |   -0.92062      -0.92000
St. dev. |   1.71186      1.51446
Skewness |   0.00000      -0.25650
Kurtosis |   0.00000      -0.26704

Chi-2 statistic:   4.033416341364921
Chi-2 p-value:     0.4015023295495953   (should be > 0.001)

How many outliers? 0

Is the sample valid? True
```

Listing 1.1: Output statistics on univariate Gaussian samples.

F.2 Multivariate tests

An example of the standard outputs of the multivariate tests in SAGA is shown in Listing 1.2.

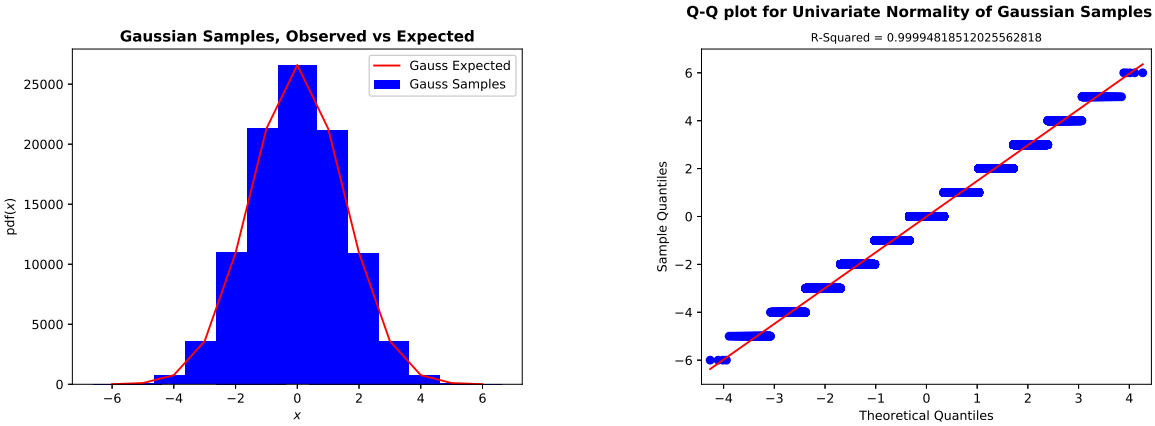
```
>>> python3              # initialize python
>>> import saga          # import the test suite
>>> sigma, data = saga.parse_multivariate_file("testdata/falcon64_avx2")
                         # parse raw file
>>> res = saga.MultivariateSamples(sigma, data)
                         # compute tests
>>> res                  # print results

Testing a centered multivariate Gaussian of dimension = 128 and sigma = 171.831
Number of samples: 63960

The test checks that the data corresponds to a multivariate Gaussian, by doing the
     following:
1 - Print the covariance matrix (visual check). One can also plot
    the covariance matrix by using self.show_covariance()).
2 - Perform the Doornik-Hansen test of multivariate normality.
    The p-value obtained should be > 0.001
3 - Perform a custom test called covariance diagonals test.
4 - Run a test of univariate normality on each coordinate

1 - Covariance matrix (128 x 128):
[[ 0.997  -0.0021  0.0065 ...  0.0014  0.0012 -0.0039]
 [-0.0021  1.0001 -0.0014 ...  0.0032  0.0005 -0.0048]
 [ 0.0065 -0.0014  1.0028 ... -0.0006  0.0074  0.0065]
 ...
 [ 0.0014  0.0032 -0.0006 ...  1.0063 -0.0022 -0.0005]
 [ 0.0012  0.0005  0.0074 ... -0.0022  0.993  -0.0008]
 [-0.0039 -0.0048  0.0065 ... -0.0005 -0.0008  1.0081]]

2 - P-value of Doornik-Hansen test:                0.2453

3 - P-value of covariance diagonals test:          0.3244

4 - Gaussian coordinates (w/ st. dev. = sigma)?    128 out of 128

>> u.univariates        # returns univariate tests on each 'row'

Testing a Gaussian sampler with center = 0 and sigma = 164.46976732471182
Number of samples: 1000

Moments  |   Expected     Empiric
---------+----------------------
Mean:    |   0.00000      -2.01000
St. dev. |   164.46977      160.92979
Skewness |   0.00000      0.01555
Kurtosis |   0.00000      -0.00831

Chi-2 statistic:   97.98787878787878
Chi-2 p-value:     0.2650065102842649   (should be > 0.001)

How many outliers? 0

Is the sample valid? True
```

Listing 1.2: Output statistics on multivariate Gaussian samples.

F.3 Supplementary, visual, 'sanity check' tests

We also provide further sanity check functionality in the code, which will be particularly useful if any of the statistical tests above fail to conform to the expected values. D'Agostino et al. [15] suggested the best way to do this is via graphical methods, thus we provide visuals for both univariate and multivariate Gaussians.

For univariate Gaussians, we provide plots of the observed and expected probability density functions (Figure 2a) and quantile-quantile (QQ) plots (Figure 2b), shown in Figure 2. The QQ plot also provides the coefficient of determination, $R^2 \in [0, 1)$, which measures how well the observed data follows the distribution we expect. There are many different errors which can be observed in QQ plots, from sampling biases to data skewness, but there are some useful guides [26], helpful websites[10][11], and online tools[12] to help analysing these plots.



(a) Observed vs Expected Gaussian PDF

(b) QQ-plot of Observed vs Expected Quantiles

Figure 2: Visual representation of the expected form of univariate Gaussian samples.

For multivariate distributions, it is typically difficult to visualise any statistical properties of a distribution beyond three dimensions. However, there is a method for checking multivariate normality using a distribution's Mahalanobis distance [42], which is a measure of the distance between certain points to a certain distribution. More specifically, it is a multi-dimensional generalisation of measuring how many standard deviations a point is away from a distribution. These sorted distances should follow a chi-square distribution [38], thus we can visualise this as a QQ-plot comparing the Mahalanobis distance versus the expected chi-square distribution. Figure 3 shows an example of our proposed multivariate normality graphical respresentation.

A final test we provide is for checking the rejection rate of the Gaussian samplers. Theoretically, the rejection rates of the Gaussian samples should decrease geometrically. An example of what the rejection rates should look like for a fixed σ are provided in Figure 4.

In order to check the rejection rates, the test suite requires the additional output of the rejection rate associated to each sample. An example of this is shown in Listings 1.3 and 1.4, which shows the typical Gaussian sampler (Listing 1.3) and the Gaussian sampler which also outputs the repetitions for each sample (Listing 1.4).

[10] https://stats.stackexchange.com/questions/101274/how-to-interpret-a-qq-plot.
[11] https://data.library.virginia.edu/understanding-q-q-plots/.
[12] https://xiongge.shinyapps.io/QQplots/

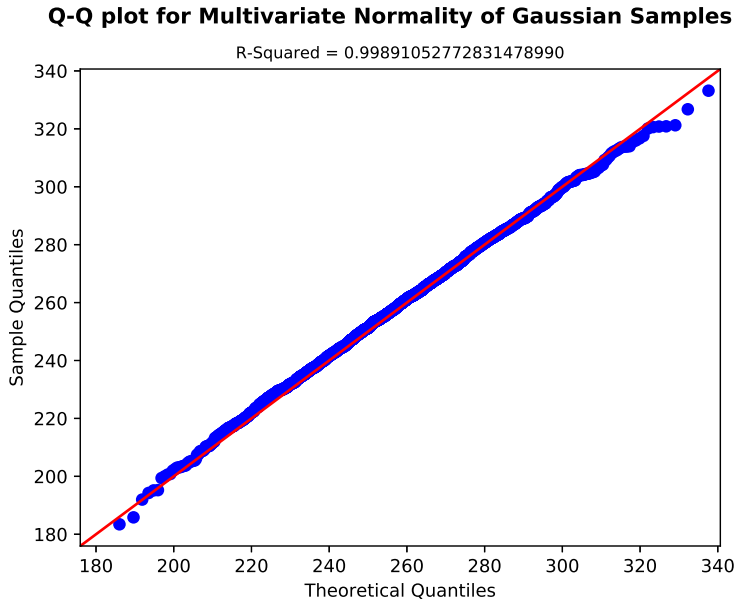


Figure 3: Visual representation of the expected form of univariate Gaussian samples.

```
def samplerz(center, sigma):
    assert(sigma < sigma0)
    assert(sigma >= sigmin)
    c0 = center - floor(center)
    sf = sigma / sigma0

    while(1):
        z0 = sampler0()
        b = randint(0, 1)
        z = ((b << 1) - 1) * z0 + b
        x = ((z - c0) ** 2) / (2 * (sigma **
            2)) - (z0 ** 2) / (2 * (sigma0 **
            2))

        if berexp(x, sf) is True:

            return floor(center) + z
```

Listing 1.3: Standard Gaussian sampler.

```
def samplerz_rep(center, sigma):
    assert(sigma < sigma0)
    assert(sigma >= sigmin)
    c0 = center - floor(center)
    sf = sigma / sigma0
    cnt = 0       # repetition counter
    while(1):
        z0 = sampler0()
        b = randint(0, 1)
        z = ((b << 1) - 1) * z0 + b
        x = ((z - c0) ** 2) / (2 * (sigma **
            2)) - (z0 ** 2) / (2 * (sigma0 **
            2))
        cnt += 1 # increment counter
        if berexp(x, sf) is True:
            # output count with value
            return floor(center) + z, cnt
```

Listing 1.4: Gaussian sampler with repetitions.

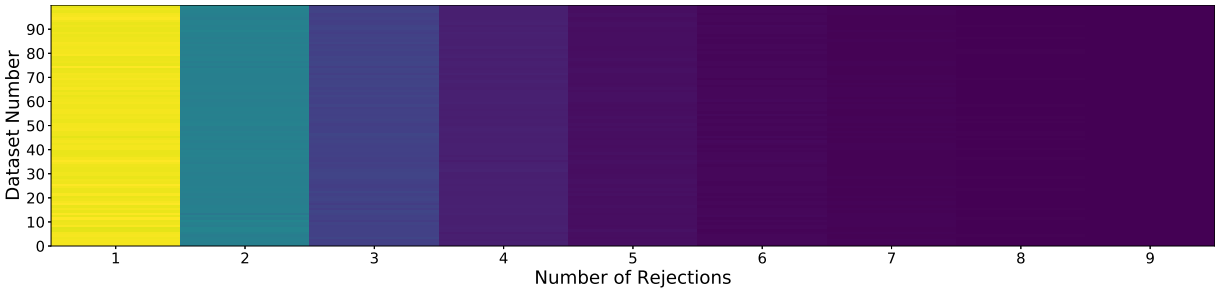


Figure 4: Expected geometric decrease in rejection numbers during Gaussian sampling.