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,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
melissa.rossi@ens.fr
[5] Inria, Paris, France

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 [9,21,49,56]. For this reason, some schemes limit or proscribe the use of Gaussians [6,36]. However, in some situations, Gaussians are unavoidable. The most prominent example is trapdoor sampling [26,40,48]: performing it with other distributions is an open question, except in limited cases [37] 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 [26,53], identity-based encryption (or IBE) [18,26], hierarchical IBE [1,11], 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, [42] implicity requires a different base sampler for each different value of σ; this limits its applicability for use cases such as Falcon [53], 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 [3,52]. 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 [46]. Randomness failures have been manually discovered and fixed in implementations of Dilithium [45], Falcon [47,51] 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 [53], however it can be used for any other scheme sampling multivariate discrete Gaussians, including but not limited to [5,12,18,25,40]. The test suite is publicly available at: https://github.com/PQShield/SAGA.

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 [42,50] proposed an arbitrary-center Gaussian sampler base, as well as a statistical tool (the max-log distance) to analyse it. [3,39,52] revisited classical techniques with the Rényi divergence. Polynomial-based methods were further studied by [4,52,60]. The use of rounded Gaussians was proposed in [31]. Knuth-Yao's DDG trees have been considered in [20,32].[2] Lazy floating-point precision was studied in [16,19]. We note that techniques dating back to von Neumann [57] allow to generate (continuous) Gaussians elegantly using finite automata [2,24,33]. While these have been considered in the context of lattice-based cryptography [15,17] they are also notoriously hard to make isochronous. Finally, [58] studied previously cited techniques with the goal of minimizing their relative error.

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

[2] We note that one could use [32] to speed up our base sampler; however this results in a huge code size (more than 50 kB). 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.

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, including a few listed below.

Lemma 1 ([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:*

- ***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 $E \subseteq Supp(\mathcal{Q})$ and $a \in (1, +\infty)$,*

$$\mathcal{Q}(E) \geq \mathcal{P}(E)^{\frac{a}{a-1}}/R_a(\mathcal{P}, \mathcal{Q}), \quad (1)$$

$$\mathcal{Q}(E) \geq \mathcal{P}(E)/R_\infty(\mathcal{P}, \mathcal{Q}). \quad (2)$$

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 2 (Lemma 3 of [52]). *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 [41] 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 2. *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

Algorithm 1. SamplerZ(σ, μ)

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

Algorithm 2. AcceptSample(σ, x)

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

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 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 $z := (2b-1) \cdot z_0 + b$ 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 [35] and accept z with probability

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

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 Sect. 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 SamplerZ(σ, μ) is then proportional to $\frac{\sigma_{\min}}{\sigma} \cdot D_{\mathbb{Z},\sigma,\mu}$ which is, after

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	$Q_{\exp}$	$Q_{\mathrm{bs}} \leq 2^{76}$

normalization exactly equal to $D_{\mathbb{Z},\sigma,\mu}$. Thus, with this construction, one can derive the following proposition.

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

In practical implementations, one cannot achieve perfect distributions. Only achieving BaseSampler $\approx D_{\mathbb{Z}^+,\sigma_{\max}}$ and 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 $D_{\mathbb{Z},\sigma,\mu}$.

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 3, 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 $Q_{\exp} = Q_{\mathrm{bs}} = \mathcal{N}_{\mathrm{iter}} \cdot Q_{\mathrm{samplZ}}$ 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 1 (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,* $\Delta\lambda := \lambda_{\mathrm{IDEAL}} - \lambda_{\mathrm{REAL}} \leq 2$.

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

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

The proof of this Theorem is given in the full version of our paper [30].

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

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

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 $P_{\text{gal}}(x) := \sum_{i=0}^{10} a_i \cdot x^i$, 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 [22] is helpful in this regard.

```
x2        ← x × x
x4        ← x2 × x2
Pgal(x)   ← (x4 × x4) × ((a8 + a9 × x) + x2 × a10)
          + (((a0 + a1 × x) + x2 × (a2 + a3 × x)) + x4 × ((a4 + a5 × x) + x2 × (a6 + a7 × x)))
```

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 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: $PDT(z) = 2^{-\theta} \lfloor 2^\theta D_{[w],\sigma_{\max}}(z) \rfloor$.
   - 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 $R_a(\mathsf{SampleCDT}_{w=19,\theta=72}, D_{\mathbb{Z}^+,\sigma_{\max}})$.

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

Algorithm 3. SampleCDT: full-table access CDT

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

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

- ◦ $\texttt{PDT}(0) = 2^{-72} \times 1697680241746640300030$
- ◦ $\texttt{PDT}(1) = 2^{-72} \times 1459943456642912959616$
- ◦$\texttt{PDT}(2) = 2^{-72} \times 928488355018011056515$
- ◦ $\texttt{PDT}(3) = 2^{-72} \times 436693944817054414619$
- ◦ $\texttt{PDT}(4) = 2^{-72} \times 151893140790369201013$
- ◦ $\texttt{PDT}(5) = 2^{-72} \times 39071441848292237840$
- ◦ $\texttt{PDT}(6) = 2^{-72} \times 7432604049020375675$
- ◦ $\texttt{PDT}(7) = 2^{-72} \times 1045641569992574730$
- ◦ $\texttt{PDT}(8) = 2^{-72} \times 108788995549429682$
- ◦ $\texttt{PDT}(9) = 2^{-72} \times 8370422445201343$
- ◦ $\texttt{PDT}(10) = 2^{-72} \times 476288472308334$
- ◦ $\texttt{PDT}(11) = 2^{-72} \times 20042553305308$
- ◦ $\texttt{PDT}(12) = 2^{-72} \times 623729532807$
- ◦ $\texttt{PDT}(13) = 2^{-72} \times 14354889437$
- ◦ $\texttt{PDT}(14) = 2^{-72} \times 244322621$
- ◦ $\texttt{PDT}(15) = 2^{-72} \times 3075302$
- ◦ $\texttt{PDT}(16) = 2^{-72} \times 28626$
- ◦ $\texttt{PDT}(17) = 2^{-72} \times 197$
- ◦ $\texttt{PDT}(18) = 2^{-72} \times 1$

Our experiment showed that for any $a \geq 509$, $R_a(\mathsf{SampleCDT}_{w=19,\theta=72}, D_{\mathbb{Z}^+,\sigma_{\max}}) \leq 1 + 2^{-80} \leq 1 + \frac{1}{4Q_{\mathrm{bs}}}$, 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 2). 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 3. *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 SamplerZ(σ, μ) follows a geometric law of parameter*

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

The proof of Lemma 3 can be found in the full version of our paper [30].

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 2. *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 2 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 1. *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 2, 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 1 can be found in the full version of our paper [30]. A nice thing about Corollary 1 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 the full version of our paper [30], 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 [48], 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 [47,51] or the famous Heartbleed (CVE-2014-0160) or ROCA vulnerabilities [44] (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 [5,12], FrodoKEM [43], identity-based encryption [10,18], and in fully homomorphic encryption [54].

7.1 Univariate Tests

The statistical tests we implement here are inspired by a previous test suite proposal called GLITCH [29]. 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 the full version of our paper [30].

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 predominantly used in other fields [13,28,38] 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 [38] and Henze-Zirkler [28] 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 [14] minimises these issues by using transformed versions of the skewness and kurtosis of the multivariate data, increasing

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

the test's power. We also note that it is much faster (essentially linear in the sample size) than [28,38] (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 the full version of our paper [30].

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 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 (Fig. 1a) instead of (scaled) identity matrices in the nominal case (Fig. 1b).

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.

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 the full version of our paper [30].

8 Application and Limitations

Our sampler has been implemented by Pornin as part of the new isochronous implementation of Falcon [51]. 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, 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

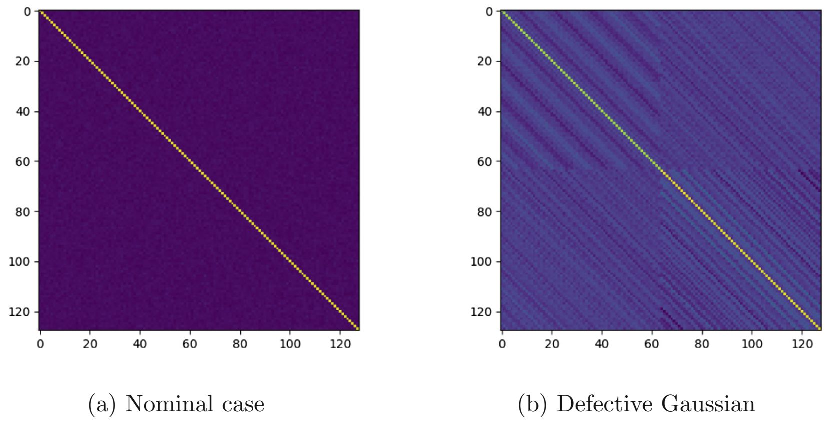

Fig. 1. Empiric covariance matrices of Falcon signatures. Figure 1a corresponds to a correct implementation of Falcon. Figure 1b corresponds to an implementation where there is a mistake when constructing the Falcon tree.

Table 2. Number of samples per second at 2.5 GHz for our sampler and [59].

Algorithm	Number of samples
This work[a]	1.84×10^6/sec
This work (AVX2)[b]	7.74×10^6/sec
[59] (AVX2)[c]	5.43×10^6/sec

[a][51] standard double-precision floating-point (IEEE 754) with SHAKE256.
[b][51] AVX2 implementation with eight ChaCha20 instances in parallel (AVX2).
[c][59] constant-time implementation with hardware AES256 (AES-NI).

limited as Cortex M4 CPUs, while some other samplers (e.g. [32] 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 [59], we scale the numbers to be based on 2.5 GHz. Note that for our sampler the number of samples per second is on average for $1.2915 < \sigma \leq 1.8502$ while for [59] $\sigma = 2$ is fixed.

In Table 3 we present the running times of the Falcon isochronous implementation [51] 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 [23], as the design *should*[5] 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, 43].

Adapting to Other Schemes. A natural question is how our algorithms could be adapted for other schemes than Falcon, for example [5, 12, 18, 25, 40]. 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 [34, 50]. It would also be interesting to see if the DDG tree-based method of [32] 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 [48] (in Falcon) entail a small constant 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 [42] and rejection sampling [59][6] 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.

[5] Compilers may alter the design, thus one should always verify the design post-compilation.

[6] The constant-time sampler in [59] may still reveal σ.

References

1. Agrawal, S., Boneh, D., Boyen, X.: Efficient lattice (H)IBE in the standard model. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 553–572. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13190-5_28
2. Ahrens, J., Dieter, U.: Extension of Forsythe's method for random sampling from the normal distribution. Math. Comput. **27**, 927–937 (1973)
3. Bai, S., Langlois, A., Lepoint, T., Stehlé, D., Steinfeld, R.: Improved security proofs in lattice-based cryptography: using the Rényi divergence rather than the statistical distance. In: Iwata, T., Cheon, J.H. (eds.) ASIACRYPT 2015, vol. 9452. LNCS, pp. 3–24. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48797-6_1
4. Barthe, G., Belaïd, S., Espitau, T., Fouque, P.A., Rossi, M., Tibouchi, M.: GALACTICS: Gaussian sampling for lattice-based constant-time implementation of cryptographic signatures, revisited. Cryptology ePrint Archive, Report 2019/511 (2019)
5. Bert, P., Fouque, P.-A., Roux-Langlois, A., Sabt, M.: Practical implementation of ring-SIS/LWE based signature and IBE. In: Lange, T., Steinwandt, R. (eds.) Post-Quantum Cryptography - 9th International Conference. PQCrypto 2018, pp. 271–291. Springer, Heidelberg (2018)
6. Bindel, N., et al.: qTESLA. Technical report, National Institute of Standards and Technology (2019). https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions
7. Bos, J.W., et al.: Frodo: take off the ring! Practical, quantum-secure key exchange from LWE. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016, pp. 1006–1018. ACM Press, October 2016
8. Breitner, J., Heninger, N.: Biased nonce sense: lattice attacks against weak ECDSA signatures in cryptocurrencies. In: Goldberg, I., Moore, T. (eds.) FC 2019. LNCS, vol. 11598, pp. 3–20. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-32101-7_1
9. Groot Bruinderink, L., Hülsing, A., Lange, T., Yarom, Y.: Flush, gauss, and reload – a cache attack on the BLISS lattice-based signature scheme. In: Gierlichs, B., Poschmann, A.Y. (eds.) CHES 2016. LNCS, vol. 9813, pp. 323–345. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53140-2_16
10. Campbell, P., Groves, M.: Practical post-quantum hierarchical identity-based encryption. In: 16th IMA International Conference on Cryptography and Coding (2017)
11. Cash, D., Hofheinz, D., Kiltz, E., Peikert, C.: Bonsai trees, or how to delegate a lattice basis. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 523–552. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13190-5_27
12. Chen, Y., Genise, N., Mukherjee, P.: Approximate trapdoors for lattices and smaller hash-and-sign signatures. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019. LNCS, vol. 11923, pp. 3–32. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-34618-8_1
13. Cox, D.R., Small, N.J.H.: Testing multivariate normality. Biometrika **65**(2), 263–272 (1978)
14. Doornik, J.A., Hansen, H.: An omnibus test for univariate and multivariate normality. Oxford Bull. Econ. Stat. **70**, 927–939 (2008)
15. Yusong, D., Wei, B., Zhang, H.: A rejection sampling algorithm for off-centered discrete Gaussian distributions over the integers. Sci. China Inf. Sci. **62**(3), 39103 (2018)

16. Ducas, L.: Signatures fondées sur les réseaux euclidiens: attaques, analyses et optimisations. Theses, École Normale Supérieure (2013)
17. Ducas, L., Durmus, A., Lepoint, T., Lyubashevsky, V.: Lattice signatures and bimodal Gaussians. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8042, pp. 40–56. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40041-4_3
18. Ducas, L., Lyubashevsky, V., Prest, T.: Efficient identity-based encryption over NTRU lattices. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014. LNCS, vol. 8874, pp. 22–41. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45608-8_2
19. Ducas, L., Nguyen, P.Q.: Faster Gaussian lattice sampling using lazy floating-point arithmetic. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 415–432. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34961-4_26
20. Dwarakanath, N.C., Galbraith, S.D.: Sampling from discrete Gaussians for lattice-based cryptography on a constrained device. Appl. Algebra Eng. Commun. Comput. **25**(3), 159–180 (2014)
21. Espitau, T., Fouque, P.A., Gérard, B., Tibouchi, M.: Side-channel attacks on BLISS lattice-based signatures: exploiting branch tracing against strongSwan and electromagnetic emanations in microcontrollers. In: Thuraisingham et al. [56], pp. 1857–1874 (2017)
22. Estrin, G.: Organization of computer systems: the fixed plus variable structure computer. In: Western Joint IRE-AIEE-ACM Computer Conference, IRE-AIEE-ACM 1960 (Western), 3–5 May 1960, pp. 33–40. ACM, New York (1960)
23. Facon, A., Guilley, S., Lec'Hvien, M., Schaub, A., Souissi, Y.: Detecting cache-timing vulnerabilities in post-quantum cryptography algorithms. In: 2018 IEEE 3rd International Verification and Security Workshop (IVSW), pp. 7–12. IEEE (2018)
24. Forsythe, G.E.: Von Neumann's comparison method for random sampling from the normal and other distributions. Math. Comput. **26**(120), 817–826 (1972)
25. Genise, N., Micciancio, D.: Faster Gaussian sampling for trapdoor lattices with arbitrary modulus. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10820, pp. 174–203. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78381-9_7
26. Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: Ladner, R.E., Dwork, C. (eds.) 40th ACM STOC, pp. 197–206. ACM Press, May 2008
27. Gilbert, H. (ed.): EUROCRYPT 2010. LNCS, vol. 6110. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13190-5
28. Henze, N., Zirkler, B.: A class of invariant consistent tests for multivariate normality. Commun. Stat.-Theory Methods **19**(10), 3595–3617 (1990)
29. Howe, J., O'Neill, M.: 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), SECRYPT, Madrid, Spain, 24–26 July 2017, vol. 4, pp. 413–419 (2017)
30. Howe, J., Prest, T., Ricosset, T., Rossi, M.: Isochronous Gaussian sampling: From inception to implementation. Cryptology ePrint Archive, Report 2019/1411 (2019)
31. Hülsing, A., Lange, T., Smeets, K.: Rounded Gaussians - fast and secure constant-time sampling for lattice-based crypto. In: Abdalla, M., Dahab, R. (eds.) PKC 2018. LNCS, vol. 10770, pp. 728–757. Springer, Heidelberg (2018)

32. Karmakar, A., Roy, S.S., Vercauteren, F., Verbauwhede, I.: 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, pp. 1–6 (2019)
33. Karney, C.F.F.: Sampling exactly from the normal distribution. ACM Trans. Math. Softw. **42**(1), 3:1–3:14 (2016)
34. Khalid, A., Howe, J., Rafferty, C., Regazzoni, F., O'Neill, M.: Compact, scalable, and efficient discrete Gaussian samplers for lattice-based cryptography. In: 2018 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1–5. IEEE (2018)
35. Lyubashevsky, V.: Fiat-Shamir with aborts: applications to lattice and factoring-based signatures. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 598–616. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-10366-7_35
36. Lyubashevsky, V., et al.: Crystals-dilithium. Technical report, National Institute of Standards and Technology (2019). https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions
37. Lyubashevsky, V., Wichs, D.: Simple lattice trapdoor sampling from a broad class of distributions. In: Katz, J. (ed.) PKC 2015. LNCS, vol. 9020, pp. 716–730. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46447-2_32
38. Mardia, K.V.: Measures of multivariate skewness and kurtosis with applications. Biometrika **57**(3), 519–530 (1970)
39. Melchor, C.A., Ricosset, T.: CDT-based Gaussian sampling: from multi to double precision. IEEE Trans. Comput. **67**(11), 1610–1621 (2018)
40. Micciancio, D., Peikert, C.: Trapdoors for lattices: simpler, tighter, faster, smaller. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 700–718. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29011-4_41
41. Micciancio, D., Regev, O.: Worst-case to average-case reductions based on Gaussian measures. SIAM J. Comput. **37**(1), 267–302 (2007)
42. Micciancio, D., Walter, M.: Gaussian sampling over the integers: efficient, generic, constant-time. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10402, pp. 455–485. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63715-0_16
43. Naehrig, M., et al.: FrodoKEM. Technical report, National Institute of Standards and Technology (2019). https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions
44. Nemec, M., Sýs, M., Svenda, P., Klinec, D., Matyas, V.: The return of coppersmith's attack: practical factorization of widely used RSA moduli. In: Thuraisingham et al. [56], pp. 1631–1648 (2017)
45. NIST et al.: Official Comment: Crystals-dilithium (2018). https://groups.google.com/a/list.nist.gov/d/msg/pqc-forum/aWxC2ynJDLE/YOsMJ2ewAAAJ
46. 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. Accessed 23 Oct 2019
47. NIST et al.: Official Comment: Falcon (bug & fixes) (2019). https://groups.google.com/a/list.nist.gov/forum/#!topic/pqc-forum/7Z8x5AMXy8s. Accessed 23 Oct 2019
48. Peikert, C.: An efficient and parallel Gaussian sampler for lattices. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 80–97. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14623-7_5

49. Pessl, P., Bruinderink, L.G., Yarom, Y.: To BLISS-B or not to be: attacking strongSwan's implementation of post-quantum signatures. In: Thuraisingham et al. [56], pp. 1843–1855 (2017)
50. Pöppelmann, T., Ducas, L., Güneysu, T.: Enhanced lattice-based signatures on reconfigurable hardware. In: Batina, L., Robshaw, M. (eds.) CHES 2014. LNCS, vol. 8731, pp. 353–370. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44709-3_20
51. Pornin, T.: New Efficient, Constant-Time Implementations of Falcon. Cryptology ePrint Archive, Report 2019/893 (2019)
52. Prest, T.: Sharper bounds in lattice-based cryptography using the Rényi divergence. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10624, pp. 347–374. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70694-8_13
53. Prest, T., et al.: FALCON. Technical report, National Institute of Standards and Technology (2019). https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions
54. Microsoft SEAL (release 3.4), October 2019. Microsoft Research, Redmond, WA. https://github.com/Microsoft/SEAL
55. Thuraisingham, B.M., Evans, D., Malkin, T., Dongyan, X. (eds.): ACM CCS 2017. ACM Press, New York (2017)
56. Tibouchi, M., Wallet, A.: One bit is all it takes: a devastating timing attack on BLISS's non-constant time sign flips. In: MathCrypt 2019 (2019)
57. von Neumann, J.: Various techniques used in connection with random digits. Natl. Bureau Standards Appl. Math Ser. **12**, 36–38 (1950)
58. Walter, M.: Sampling the integers with low relative error. In: Buchmann, J., Nitaj, A., Rachidi, T. (eds.) AFRICACRYPT 2019. LNCS, vol. 11627, pp. 157–180. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-23696-0_9
59. Zhao, R.K., Steinfeld, R., Sakzad, A.: Compact and scalable arbitrary-centered discrete Gaussian sampling over integers. Cryptology ePrint Archive, Report 2019/1011 (2019)
60. Zhao, R.K., Steinfeld, R., Sakzad, A.: Facct: fast, compact, and constant-time discrete Gaussian sampler over integers. IEEE Trans. Comput. (2019)