Simple, Fast and Constant-Time Gaussian Sampling over the Integers for Falcon

Thomas Prest[1], Thomas Ricosset[2], and Mélissa Rossi[2,3]

[1] PQShield
thomas.prest@pqshield.com
[2] Thales
thomas.ricosset@thalesgroup.com
[3] École normale supérieure, CNRS, PSL University, Inria, Paris, France
melissa.rossi@ens.fr

Abstract. We propose a constant-time implementation of the signature procedure of Falcon. Doing this boils down to proposing a generic and constant-time Gaussian sampler over the integers. We show how to do so as efficiently as possibly while preserving correctness. We formally prove that the resulting sampler is constant-time. Finally, we implement our sampler in Falcon; the efficiency loss in the resulting implementation is reasonably low compared to the non constant-time, reference implementation, and even with this penalty, Falcon remains one of the fastest signature scheme candidates.

Keywords: Gaussian sampling; constant-time algorithms; lattice-based cryptography; implementation.

1 Introduction

As of the round 2 of NIST's PQC standardisation process, Falcon is one of the 9 signature schemes still considered for standardisation. One might note that, in contrast to some other signature schemes (e.g. Dilithium [LDK+19], LUOV [BPSV19], SPHINCS$^+$ [HBD+19]), the reference implementation of Falcon is currently not constant-time. While no attack has exploited this fact so far, the cautionary tale of BLISS [DDLL13] has shown that a lattice-based scheme can be perfectly secure in a black-box model, yet fall prey to numerous side-channel attacks if unprotected [EFGT17,PBY17,BDE+18,BBE+19]. As a first step, it is therefore highly desirable to have a constant-time implementation of Falcon.

The goal of this work is move towards a portable and constant-time implementation of Falcon. We focus our scope on the signature generation, and show that it is constant-time if and only if the Gaussian sampling over the integers (a building block of Falcon) is constant-time. Therefore, our goal is equivalent to proposing a constant-time Gaussian sampler.

Since Falcon was first proposed [PFH+17], several works have greatly improved the state of the art regarding constant-time Gaussian sampling. We consider in particular two of these works, FACCT [ZSS18] and GALACTICS [BBE+19], for tackling the rejection sampling step. Our contributions are as follows.

- We show that given two algorithms approximating a centered, fixed-deviation Gaussian sampler (the base sampler) and a rejection sampling algorithm, respectively, one can build a Gaussian sampler with arbitrary center and standard deviation in black-box (thus modularly). We show that only mild requirements are required on the correctness of these two building blocks to ensure the correctness of the complete sampler. This is coinjointly due to NIST's limitation on the number of signatures ($Q_s \leq 2^{64}$) and the use of the Rényi divergence in our security proofs.
- Recent works [ZSS18,KRVV19] have observed that even if the building blocks (base sampler and rejection sampling) are constant-time, it is still possible that the total number of iterations of the complete sampler may leak the private key. Whether that was the case in Falcon was left as an open question by both works. In this work, we answer that question and show that the number of iterations of our sampler does not leak the private key. Interestingly, our proofs rely extensively on the smoothing parameter. Originally, this quantity plays an important role in showing that Falcon is secure in a black-box model. Somewhat surprisingly, it is also crucial in making our sampler constant-time.

– We propose an instantiation of our sampler using a small cumulative distribution table for the base sampler, and either FACCT [ZSS18] or GALACTICS [BBE+19] for the rejection sampling. A few observations allow us to use only 70 bits of randomness for the base sampler and (on average) 8 bits for rejection sampling. We want to emphasize that this instantiation is extremely simple. On the other hand, since our design is modular, it does leave plenty of room to propose improvements to either of the building blocks, or to their combination. We describe a few of these potential improvements as well.
– We propose a constant time implementation of Falcon using our constant-time sampler and provide comparisons with other implementations on an Intel Core i7-6500U CPU (clocked at 2.5 GHz). Compared to the reference implementation of Falcon (which is not constant-time), ours is slightly slower, but it still computes thousands of signatures per second. It is also faster than the reference implementation of Dilithium [LDK+19].

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 [PDG14], Micciancio and Walter [MW17] proposed an arbitrary-center Gaussian sampler base, as well as a statistical tool (the max-log distance) to analyse it. Prest [Pre17] revisited techniques based on precomputed tables or polynomial interpolation, along with an improved security analysis using the Rényi divergence (building upon work by Bai et al. [BLL+15].). Polynomial-based methods were further studied by Zhao et al. [ZSS18] and Barthe et al.[BBE+19]. Hülsing et al. [HLS18] proposed the use of rounded Gaussians. Dwarakanath and Galbraith [DG14] proposed to use Knuth-Yao's discrete distribution generating (DDG) trees; an optimized and constant-time instantiation of this idea has recently been proposed by Karmakar et al. [KRVV19]. Ducas and Nguyen [DN12] proposed a variant of rejection sampling using only standard double precision floating-point numbers in most cases, and this work was extended in Ducas' PhD thesis [Duc13]. It is to be noted that techniques first proposed by von Neumann [von50] allow to generate (continuous) Gaussians simply and elegantly by combining rejection sampling and finite automata [For72,AD73,Kar16]. While these techniques have been considered in the context of lattice-based cryptography [DDLL13,DWZ18] they are also notoriously hard to make constant-time. Finally, Walter [Wal19] studied a few of the 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 $D_{S,\sigma,\mu}(z) = \rho_{\sigma,\mu}(z)/\rho_{\sigma,\mu}(S)$.

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 a few properties useful for cryptography, including a probability preservation property with dramatically different implications than the one verified by the statistical distance.

Lemma 1 (Lemma 2.9 of [BLL+15]). *For two distributions $\mathcal{P}, \mathcal{Q}$ and two families of distributions $(\mathcal{P}_i)_i, (\mathcal{Q}_i)_i$, the Rényi divergence verifies the following 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 $E \subseteq \mathrm{Supp}(\mathcal{Q})$ and $a \in (1, +\infty)$,*

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

- ***Weak triangle inequality.*** *For $P_1 P_2, P_3$ with $\mathrm{Supp}(P_i) \subseteq \mathrm{Supp}(P_{i+1})$:*

$$R_a(P_1, P_3) \leq \begin{cases} R_a(P_1, P_2) \cdot R_\infty(P_2, P_3), \\ R_\infty(P_1, P_2)^{\frac{a}{a-1}} \cdot R_a(P_2, P_3) \quad \text{if } a \in (1, +\infty). \end{cases}$$

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

Lemma 2 (Lemma 3 of [Pre17]). *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 $\sqrt{2\pi}$. The following bound is given by [MR07]:

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

3.4 Constant-time algorithms

We now give a semi-formal definition of constant-time algorithms.

Definition 2. *Let $\mathcal{A}$ be an (probabilistic or deterministic) algorithm. $\mathcal{A}$ is said to be constant-time if its execution time is independent of its input and output. In addition, $\mathcal{A}$ is still considered to be constant-time with the following relaxations:*

- *If a subset $\mathcal{I}$ (resp. $\mathcal{O}$) of the input (resp. output) is tagged as sensitive, $\mathcal{A}$ is still considered as constant-time if its running time is independent of the values taken by $\mathcal{I}$ (resp. $\mathcal{O}$).*
- *If there exists a distribution $\mathcal{D}$ independent of the (sensitive) input/output such that the running time of $\mathcal{A}$ is negligibly close (for a clearly identified divergence) to $\mathcal{D}$.*

In addition, we may extend the notion of being "constant-time" to algorithmic objects such as conditional loops: if an algorithm contains a conditional loop which loops over a number of time which is independent of sensitive values, we abusively say that this loop is "constant-time".

It is clear that the composition of constant-time elements is constant-time:

- If $\mathcal{A}, \mathcal{B}$ are constant-time, then the successive execution $\{\mathcal{A}; \mathcal{B}; \}$ of $\mathcal{A}$ then $\mathcal{B}$ is constant-time;
- In addition, let $\mathbf{loop}(\cdot)$ be the process of executing repeatedly an element a variable number of times, if $\mathcal{A}, \mathbf{loop}(\cdot)$ are constant-time, then $\mathbf{loop}(\mathcal{A})$ is constant-time.

We observe that saying "$\mathcal{A}$ is a constant-time algorithm" is different from saying "the running time of $\mathcal{A}$ is constant". While the latter certainly implies the former, our definition is sufficient for cryptographic purposes and is much easier to achieve.

3.5 (Non) Constant-Time Operations in Falcon

We now study which parts of Falcon are constant-time and which are not. Our key takeaway is that Falcon is constant-time if and only if the Gaussian sampler over $\mathbb{Z}$ is constant-time.

Scope

In Falcon as in any signature scheme, two algorithms can leak the secret key: the key generation algorithm and the signing algorithm. This work focuses on the signing algorithm, as the number of signature queries per private key can be high (up to 2^{64} as par [NIS16]), whereas the key generation algorithm is typically only executed once per private key. We note that making the key-gereration constant-time is easy and does not have a huge impact on performances.

Breakdown of the signing procedure

In Figure 1, we show the call graph of the signing algorithm of Falcon [PFH+19]. The subroutine HashToPoint involves only the message and is therefore irrelevant to our analysis; similarly, Compress involves only the signature and can be discarded.

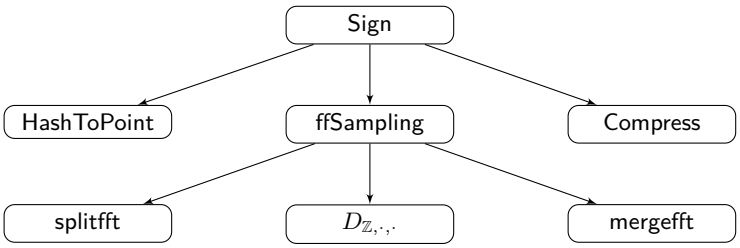

Fig. 1. Flowchart of the signature

Putting aside the sampling over $\mathbb{Z}$, all the operations of Sign, ffSampling, splitfft and mergefft involve only elementary operations over (complex) floating-point numbers: addition, subtraction, multiplication and division. In all the division operations, the divisor is either a public constant or an element of sk and can therefore be precomputed at a small cost (naively, this at most doubles the size of sk).

We assume that addition, subtraction and multiplication are constant-time; whether this is the case is a complex issue (depending on the architecture, the compiler, etc.), we note that this is true for most recent Intel and ARM processors[4]. We also note that as per [PFH+19], Falcon does not require the support of subnormals, infinites and NaNs. This makes it easier to write dedicated, constant-time assembly code if the target platform does no natively implements constant-time addition, subtraction and multiplication.

[4]See https://www.agner.org/optimize/instruction_tables.pdf for a comprehensive breakdown.

Falcon's Gaussian sampler

When signing a message, Falcon calls twice an arbitrary Gaussian sampler over the integers for each leaf of the LDL tree. Each call should produce a sampler from $D_{\sigma,\mu}$. The centers μ change from call to call, and are dynamically computed based on the message to sign, and the values returned by previous calls to the sampler. The values of σ depend on the private key, but not on the message. In the Falcon reference code, rejection sampling with regards to a bimodal Gaussian is used:

1. The target μ is moved into the $[0, 1)$ interval by adding an appropriate integer value, which will be subtracted from the sampling result at the end. For the rest of this paper, we assume that $\mu \in [0, 1)$.
2. A non-negative integer z_0 is sampled from a half- Gaussian distribution $D_{\mathbb{Z}^+,\sigma_0}$, with σ_0 a fixed constant.
3. A random bit b is obtained, to compute $z = (2b - 1) \cdot z_0 + b$. The integer z follows a bimodal Gaussian distribution $G_{\mathbb{Z},\sigma_0}$.[5]
4. Finally, rejection sampling is applied: z follows the distribution $G_{\mathbb{Z},\sigma_0}(z) = \frac{\rho_{\sigma_0}(z-b)}{\rho_{\sigma_0}(\mathbb{Z})}$. One thus returns z with probability proportional to $\frac{D_{\sigma,\mu}(z)}{G_{\mathbb{Z},\sigma_0}(z)}$; otherwise, one starts over.

The Falcon reference code uses random values obtained from ChaCha20 and standard double precision floating-point values, which is sufficient to achieve the required security levels from a Rényi argument. It is worth noting that the Gaussian sampler in the Falcon reference code is not constant time, therefore it may be a source of leakage for a side-channel attack. In particular, it uses a lazy CDT approach to sample from the half Gaussian distribution whose execution time is heavily dependent on its output.

4 Our Sampler

We now describe our constant-time sampling algorithm which, given a fixed somewhat small $\sigma_0 \in \mathbb{R}^\star_+$, samples the distribution $D_{\mathbb{Z},\sigma,\mu}$ for any $\sigma \leq \sigma_0$ and any $\mu \in [0, 1]$. This is done by instantiating the rejection sampling method to sample from the target probability distribution $D_{\mathbb{Z},\sigma,\mu}$, given a source bound to the bimodal Gaussian proposal distribution $G_{\mathbb{Z},\sigma_0}$. At this point we will define the notation:

$$\nu_\sigma(x) := \begin{cases} \rho_{\sigma_0}(x) & \text{if } x \leq 0, \\ \rho_{\sigma_0}(x-1) & \text{if } x \geq 1. \end{cases} \tag{4}$$

The dominating density $G_{\mathbb{Z},\sigma_0}(z) := \frac{\nu_{\sigma_0}(z)}{\nu_{\sigma_0}(\mathbb{Z})}$ for all $z \in \mathbb{Z}$ is proportional to $D_{\mathbb{Z},\sigma_0}(z)$ over the negative integers and to $D_{\mathbb{Z},\sigma_0,1}(z)$ over the non-zero positive integers. One can notice that due to the second mode of $\nu_{\sigma_0}(x)$ which adds a $\rho_{\sigma_0}(0) = 1$ term to the normalizing constant, we have

$$\nu_{\sigma_0}(\mathbb{Z}) = \rho_{\sigma_0}(\mathbb{Z}) + 1 = 2 \cdot \rho_{\sigma_0}(\mathbb{Z}^+).$$

Figure 2 illustrates the proposal distribution $G_{\mathbb{Z},\sigma_0}$ for $\sigma_0 = 2$, and some possible target distributions $D_{\mathbb{Z},\sigma,\mu}$. We note that for σ close to σ_0 the distance between proposal and target curves is small with regard to the range of μ, which leads to a small rejection rate.
The sampling method works as follows:

1. Generating a sample from a bimodal Gaussian $G_{\mathbb{Z},\sigma_0}$ which is as close as possible to any target Gaussian $D_{\mathbb{Z},\sigma,\mu}$. This is studied in section 4.1 and can be done during an *offline* phase independent of σ and μ.
2. Performing rejection sampling over the sample generated to obtain $D_{\mathbb{Z},\sigma,\mu}$. This is studied in section 4.2 and have to be done during the *online* phase due to the dependence of σ and μ.

Our arbitrary sampler combines the steps 1 and 2 in a sensible way to obtain $D_{\mathbb{Z},\sigma,\mu}$; this is studied in section 4.3.

[5] To be exact, it is an bimodal half-Gaussian.

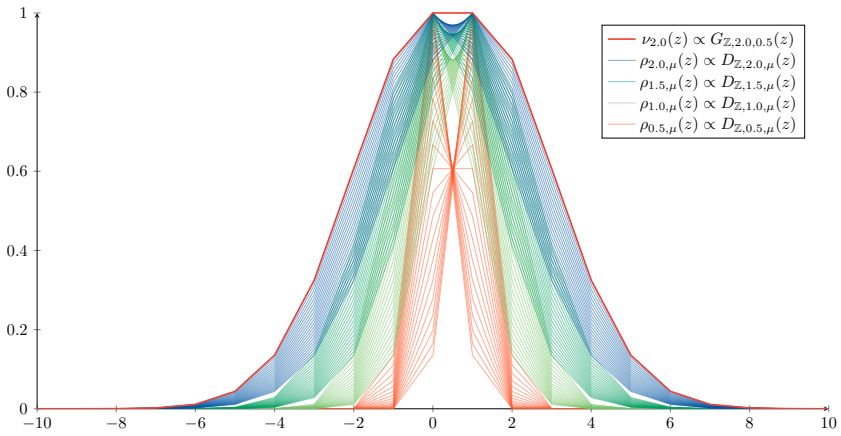

Fig. 2. The dominating curve $\nu_{2.0}$ proportional to the proposal density and the target curves for some distributions $D_{\mathbb{Z},\sigma\in[1/2,2],\mu\in[0,1]}$.

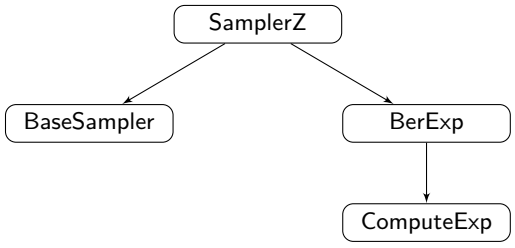

Fig. 3. Flowchart of our sampler

4.1 Sampling from the Bimodal Gaussian

We sample from the bimodal Gaussian $G_{\mathbb{Z},\sigma_0}$ by drawing $z_0 \leftarrow D_{\mathbb{Z}^+,\sigma_0}$, $b \leftarrow \{0,1\}$ uniformly, and $z \leftarrow (2b-1)\cdot z_0$. Hence:

$$G_{\mathbb{Z},\sigma_0}(z) = \frac{1}{2}\begin{cases} D_{\mathbb{Z}^+,\sigma_0}(-z) & \text{if } z \le 0 \\ D_{\mathbb{Z}^+,\sigma_0}(z-1) & \text{if } z \ge 1 \end{cases} \tag{5}$$

In practice, we will sample from $D_{\mathbb{Z}^+,\sigma_0}$ using an algorithm BaseSampler. The choice of BaseSampler depends on many external – and sometimes conflicting – constraints: speed of operations, constant-time, memory access, randomness, etc. For clarity, we will describe only the requirements on the base sampler in section 4.3 to maintain the security of our arbitrary sampler. Concrete instantiations of BaseSampler are proposed in Section 6.

4.2 BerExp: Rejection Sampling

In the rejection sampling step, we accept the proposal sample $z = (2b-1)\cdot z_0 + b$ with a probability $P_{\mathsf{accept}}(z)$ to obtain the target distribution $D_{\sigma,\mu}$. We take P_{accept} as follows:

$$\begin{aligned} P_{\mathsf{accept}}(z) &= \frac{\rho_{\sigma,\mu}(z)}{\nu_{\sigma_0}(z)} \\ &= \begin{cases} \exp\left(\frac{z^2}{2\sigma_0^2} - \frac{(z-\mu)^2}{2\sigma^2}\right) & \text{if } z \le 0 \\ \exp\left(\frac{(z-1)^2}{2\sigma_0^2} - \frac{(z-\mu)^2}{2\sigma^2}\right) & \text{if } z \ge 1 \end{cases} \\ &= \exp\left(\frac{z_0^2}{2\sigma_0^2} - \frac{(z-\mu)^2}{2\sigma^2}\right). \end{aligned}$$

We use a Bernoulli sampler BerExp_C, presented in Algorithm 1 which samples from the Bernoulli distribution with parameter $C \cdot \exp(-x)$ for any $x \in \mathbb{R}^+$. The value C is a scaling factor which purpose is explained in Section 6.3.

Algorithm 1 $\mathsf{BerExp}_C(x)$

Require: $x > 0$, $C \in (0, 1]$
Ensure: $b \sim \mathcal{B}_{C \cdot \exp(-x)}$
1: $p \leftarrow C \cdot \mathsf{ComputeExp}(-x)$
2: $i \leftarrow 1$
3: **do**
4: $\quad i \leftarrow i \cdot 2^8$
5: $\quad u \leftarrow [\![0, 2^8 - 1]\!]$ uniformly
6: $\quad v \leftarrow \lfloor p \cdot i \rfloor$ `& 0xff`
7: **while** $u = v$
8: **return** $(u < v)$

If $\mathsf{ComputeExp}$ perfectly computes $\exp(-x)$, the correctness of $\mathsf{BerExp}_C(x)$ directly follows the fact that for any $p \in [0, 1]$ and for u drawn uniformly at random in $[0, 1]$, the probability to have $u < p$ is equal to p. However, that assumption is not necessarily true in practice, where algorithms perform computations over real numbers with a high but finite *precision*. We refer to Section 5 for the quantification of the security loss induced by the use of an approximate exponential computation.

4.3 SamplerZ: The Full Sampler

Our constant-time sampler, formally described in Algorithm 2, works as follows:

1. Use $\mathsf{BaseSampler}$ to generate a sample z_0 from $D_{\mathbb{Z}^+,\sigma_0}$.
2. Sample a random bit b, and compute $z = (2b - 1) \cdot z_0 + b$.
3. Call BerExp_C to determine if z is returned or rejected, start again if necessary.

Algorithm 2 $\mathsf{SamplerZ}(\sigma, \mu)$

Require: $\mu \in [0, 1)$, $\sigma \leq \sigma_0$, a scaling factor $C = C(\sigma) \in (0, 1]$
Ensure: $z \sim D_{\mathbb{Z},\sigma,\mu}$
1: **while** True **do**
2: $\quad z_0 \leftarrow \mathsf{BaseSampler}()$
3: $\quad b \leftarrow \{0, 1\}$ uniformly
4: $\quad z \leftarrow (2b - 1) \cdot z_0 + b$
5: $\quad x \leftarrow \frac{(z-\mu)^2}{2\sigma^2} - \frac{z_0^2}{2\sigma_0^2}$
6: $\quad$ **if** $\mathsf{BerExp}_{C(\sigma)}(x)$ **then**
7: $\quad\quad$ **return** z

Here C actually depends on σ. It is a scaling function mapping σ to a value in $(0, 1]$. It serves no functional purpose: the algorithm will be correct for any choice of C (including the simplest case $C = 1$). However we will see in Section 6.3 that adequately choosing C makes $\mathsf{SamplerZ}$ fully constant-time.

5 Security proof for SamplerZ

In this section, we quantify the security loss incurred when an implementation of Falcon using an ideal sampler algorithm $D_{\mathbb{Z},\sigma,\mu}$ is replaced by an implementation calling the algorithm $\mathsf{SamplerZ}$ instead. In order to do this, we first define requirements on the building blocks $\mathsf{ComputeExp}$ and $\mathsf{BaseSampler}$, then we quantify

the bit security loss.

Beforehand, Table 1 gives the considered maximum number of calls to SamplerZ, BaseSampler and ComputeExp. Due to the rejection sampling, there will be a (potentially infinite) number of iterations of the **while** loop. We note $\mathcal{N}_{\text{iter}}$ the expected value of the number of iterations. By a classical central limit argument, the number of calls to BaseSampler and ComputeExp will only be marginally higher than $\mathcal{N}_{\text{iter}}$ times the number of calls to SamplerZ. In Section 6, we show that $\mathcal{N}_{\text{iter}} \leq 2$.

Table 1. Number of calls to SamplerZ, BaseSampler and ComputeExp

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

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

1. (Ideal) The ideal $D_{\mathbb{Z},\sigma,\mu}$ is called.
2. (Inter) A hybrid version of SamplerZ where ComputeExp outputs the exact exp value is called.
3. (Real) SamplerZ is called.

Let λ be the security parameter of the Real case and $a := 2\lambda + 1$; 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. In a first step, we compute the security loss as a function of $R_a(\text{Real}, \text{Inter})$ and $R_a(\text{Inter}, \text{Ideal})$ in Theorem 1. In a second step, we define the requirements on ComputeExp and BaseSampler to ensure suitable values for $R_a(\text{Real}, \text{Inter})$ and $R_a(\text{Inter}, \text{Ideal})$.

Theorem 1. *Let λ be the bit security of the* Real *case. Then the bit security of the* Ideal *case is upper bounded by:*

$$\lambda + \log_2 \left(\sqrt{2}^{\frac{a}{a-1}+1} \cdot R_a(\text{Real}, \text{Inter})^{\frac{aQ_{\text{exp}}}{a-1}} \cdot R_a(\text{Inter}, \text{Ideal})^{Q_{\text{bs}}} \right)$$

where $a = 2\lambda + 1$.

Proof. 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. By definition, $\delta_{\text{Real}} \geq 2^{-\lambda}$. By Lemma 1, (1):

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

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

Taking $a := 2\lambda + 1$, 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,

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

which concludes the proof. □

The following corollary provides concrete requirements on the Rényi divergence terms.

Corollary 1 (Security loss). *If* $R_a(\textsc{Real}, \textsc{Inter}) \leq 1 + \frac{a-1}{4aQ_{\exp}}$ *and* $R_a(\textsc{Inter}, \textsc{Ideal}) \leq 1 + \frac{1}{4Q_{\text{bs}}}$*, at most* 2 *bits of security are lost by using* SamplerZ *instead of the ideal distribution* $D_{\mathbb{Z},\sigma,\mu}$ *in* Falcon*.*

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

$$R_a(\textsc{Real}, \textsc{Inter})^{\frac{aQ_{\exp}}{a-1}} \leq \exp(1/4) \leq \sqrt{2},$$
$$R_a(\textsc{Inter}, \textsc{Ideal})^{Q_{\text{bs}}} \leq \sqrt{2}.$$

Thus the security loss is upper bounded by $\log_2\left(\sqrt{2}^{\frac{a}{a-1}+1} \cdot 2\right) = \frac{\frac{a}{a-1}+3}{2} \leq 2$. □

5.1 Requirements on ComputeExp

In this section, we focus on the first requirement of Corollary 1.

Lemma 3 (Requirement on ComputeExp). *Recall that* $a = 2\lambda + 1$ *where* λ *is the security parameter in the* Real *case. The replacement of* ComputeExp *by a polynomial* P *such that*

$$\forall x \in \text{Supp}(\textit{ComputeExp}), \left|\frac{P(x) - \exp(x)}{\exp(x)}\right| \leq \sqrt{\frac{a-1}{2 \cdot a^2 \cdot Q_{\exp}}} \tag{6}$$

ensures $R_a(\textsc{Real}, \textsc{Inter}) \leq 1 + \frac{a-1}{4aQ_{\exp}}$.

Proof. Suppose that (6) is true. Then, $\forall x \in \text{Supp}(\mathsf{ComputeExp})$:

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

A straightforward application of Lemma 2 yields the result. □

Assuming that 256 bits are claimed in the Ideal case, then $\lambda = 254$ and $\sqrt{\frac{a-1}{2 \cdot a^2 \cdot Q_{\exp}}} \approx 2^{-43.5}$. In the following, we depict possible choices for polynomials to replace ComputeExp that verify Lemma 3. For efficiency reasons, we reduce the interval of for the polynomial approximation to $[0, \ln(2)]$. Indeed, 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.

Floating point version. One can directly replace ComputeExp by the polynomial approximation denoted P_{fp} provided in FACCT [ZSS18]. The article [ZSS18] actually provides a polynomial approximation for $x \mapsto 2^x$ and not $\exp(\cdot)$. However, the associated implementation files provide a polynomial approximation P_{facct} specially computed for Falcon.[6] It verifies:

$$\forall x \in [0, \ln(2)], \left|\frac{P_{\text{facct}}(x) - \exp(x)}{\exp(x)}\right| \leq 2^{-45}.$$

Fixed-point version. For a portable version using fixed precision arithmetics, one could use the tool provided in GALACTICS [BBE+19]. This tool generates polynomial approximations that allow a computation in fixed precision with chosen size of coefficients and degree. One can generate a polynomial, denoted P_{gal}, to approximate the function $\exp(\cdot)$ on $[0, \ln(2)]$ that verifies

$$\forall x \in [0, \ln(2)] \left|\frac{P_{\text{gal}}(x) - \exp(x)}{\exp(x)}\right| \leq 2^{-44}.$$

As an example, for 32-bit coefficients and a degree 10, GALACTICS tool provides this polynomial:

$$P_{\text{gal}}(x) = \sum_{i=0}^{10} a_i \cdot x^i,$$

with:

[6] https://github.com/raykzhao/gaussian/blob/master/applications/falcon/fpr-double.h

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

For any $x \in [0, \ln(2)]$, P_{gal} verifies $\left|\frac{P_{\text{gal}}(x) - \exp(x)}{\exp(x)}\right| \leq 2^{-47}$, which is sufficient as per Lemma 3 to ensure security. We now discuss two methods for evaluating efficiently P_{gal}: Horner's method and Estrin's method.

Horner's form. A natural way to evaluate P_{gal} efficiently is by writing it in Horner's form:

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

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

$$\begin{aligned}
&\mathtt{y} \leftarrow \mathtt{a}_{10} \\
&\mathtt{y} \leftarrow \mathtt{a}_9 + \mathtt{y} \times \mathtt{x} \\
&\quad \vdots \\
&\mathtt{y} \leftarrow \mathtt{a}_1 + \mathtt{y} \times \mathtt{x} \\
&\mathtt{y} \leftarrow \mathtt{a}_0 + \mathtt{y} \times \mathtt{x}
\end{aligned}$$

Estrin's form. Some architectures enjoy some level of parallelism, in which case it is desirable to minimise the depth of the circuit computing P_{gal}.[7] Writing P_{gal} in Estrin's form [Est60] is helpful in this regard:

$$\begin{aligned}
\mathtt{x}_2 \quad &\leftarrow \mathtt{x} \times \mathtt{x} \\
\mathtt{x}_4 \quad &\leftarrow \mathtt{x}_2 \times \mathtt{x}_2 \\
\mathtt{P}_{\text{gal}}(\mathtt{x}) &\leftarrow (\mathtt{x}_4 \times \mathtt{x}_4) \times ((\mathtt{a}_8 + \mathtt{a}_9 \times \mathtt{x}) + \mathtt{x}_2 \times \mathtt{a}_{10}) \\
&\quad + (((\mathtt{a}_0 + \mathtt{a}_1 \times \mathtt{x}) + \mathtt{x}_2 \times (\mathtt{a}_2 + \mathtt{a}_3 \times \mathtt{x})) + \mathtt{x}_4 \times ((\mathtt{a}_4 + \mathtt{a}_5 \times \mathtt{x}) + \mathtt{x}_2 \times (\mathtt{a}_6 + \mathtt{a}_7 \times \mathtt{x})))
\end{aligned}$$

We are currently investigating several tradeoffs for P_{gal}. Distinct platforms may lead to different tradeoffs between size, degree, etc. Adding ad-hoc tweaks like reducing the size of the interval also impacts the performance. In addition, parallelizability and the size of registers vary between platforms.

5.2 Requirements on BaseSampler

Lemma 4. *For $\bar{D}$ denoting the output distribution of BaseSampler,*

$$R_a(\textsc{Inter}, \textsc{Ideal}) \leq R_a(\bar{D}, D_{\mathbb{Z}^+,\sigma_0}).$$

Proof. Let $\bar{G}$ be the distribution of z before rejection sampling (Step 4). All the online phase consists in multiplying the output distribution by $z \mapsto \exp\left(\frac{(z-\mu)^2}{2\sigma^2} - \frac{z_0^2}{2\sigma_0^2}\right)$. By data processing, we get

$$R_a(\textsc{Inter}, \textsc{Ideal}) \leq R_a(\bar{G}, G_{\mathbb{Z},\sigma_0,1/2})$$

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

$$R_a(\bar{G}, G_{\mathbb{Z},\sigma_0,1/2}) = R_a((2b-1)\bar{D} + b, (2b-1)D_{\mathbb{Z}^+,\sigma_0} + b),$$

we re-apply data processing and obtain $R_a(\textsc{Inter}, \textsc{Ideal}) \leq R_a(\bar{D}, D_{\mathbb{Z}^+,\sigma_0})$. □

With Lemma 4, we proved that requiring $R_a(\bar{D}, D_{\mathbb{Z}^+,\sigma_0}) \leq 1 + \frac{1}{4Q_{\text{bs}}}$ along with $R_a(\textsc{Real}, \textsc{Inter}) \leq 1 + \frac{a-1}{4aQ_{\text{exp}}}$ is enough to satisfy Corollary 1. Thus, we require that $R_a(\mathsf{BaseSampler}, D_{\mathbb{Z}^+,\sigma_0}) \leq 1 + \frac{1}{4Q_{\text{bs}}}$.

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

Candidates for BaseSampler In the sequel, we discuss the possible options for BaseSampler that satisfy the requirements of Lemma 4. Several constant-time samplers exist for centered Gaussians with fixed standard deviation. We present two choices.

Portable option This simple option relies on a cumulative distribution table (CDT). We precompute a table of the cumulative distribution function of $D_{\mathbb{Z}^+,\sigma_0}$ 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, this can be done relatively 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 respectivelly the number of elements of the CDT and the precision of its coefficients.

Algorithm 3 $\mathsf{BaseSampler}_{w,\theta}$: 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**
  $b \leftarrow (\mathrm{CDT}[w] \geq u)$ ▷ $b = 1$ if it is true and 0 otherwise
  $z \leftarrow z + b$
return z

We use a simple script that, given σ_0 and θ as inputs:

1. Compute the smallest tailcut w such that the Renyi divergence R_a between the ideal distribution $D_{\mathbb{Z}^+,\sigma_0}$ and its restriction to $\{0,\ldots,w\}$ (noted $D_{[w],\sigma_0}$) verifies $R_a(D_{[w],\sigma_0}, D_{\mathbb{Z}^+,\sigma_0}) \leq 1 + \frac{1}{4Q_{\mathrm{bs}}}$;
2. Rounds the CDT of $D_{[w],\sigma_0}$ with θ bits of absolute precision. This rounding is done "cleverly":
   - for $z \leq 1$, the value $D_{[w],\sigma_0}(z)$ is truncated: $P(z) = 2^{-\theta}\left\lfloor 2^{\theta} D_{[w],\sigma_0}(z)\right\rfloor$.
   - in order to have a probability distribution, $P(0) = 1 - \sum_{z\geq 1} P(z)$.

   It is then easy to show that $R_\infty(P, D_{[w],\sigma_0}) = \frac{P(0)}{D_{[w],\sigma_0}(0)} \leq \frac{2^{-\theta}w}{D_{[w],\sigma_0}(0)} \approx \frac{2^{-\theta}w}{\sigma_0\sqrt{\pi/2}+1}$.

At that point, one can use the weak triangle inequality of Lemma 1. Interestingly, and for reasons which we did not pinpoint, some values of σ_0 yield in practice even better bounds (by a few bits of security) for the Rényi divergence than predicted by the theory. Indeed, taking $\sigma_0 = 1.8205$ and $\theta = 72$, we have:

- $\mathtt{P(0)} = 2^{-72} \times \mathtt{169768024174664030030}$
- $\mathtt{P(1)} = 2^{-72} \times \mathtt{145994345664291295961 6}$
- $\mathtt{P(2)} = 2^{-72} \times \mathtt{92848835501801105651 5}$
- $\mathtt{P(3)} = 2^{-72} \times \mathtt{43669394481705441461 9}$
- $\mathtt{P(4)} = 2^{-72} \times \mathtt{15189314079036920101 3}$
- $\mathtt{P(5)} = 2^{-72} \times \mathtt{3907144184829223784 0}$
- $\mathtt{P(6)} = 2^{-72} \times \mathtt{7432604049020375675}$
- $\mathtt{P(7)} = 2^{-72} \times \mathtt{1045641569992574730}$
- $\mathtt{P(8)} = 2^{-72} \times \mathtt{108788995549429682}$
- $\mathtt{P(9)} = 2^{-72} \times \mathtt{8370422445201343}$
- $\mathtt{P(10)} = 2^{-72} \times \mathtt{476288472308334}$
- $\mathtt{P(11)} = 2^{-72} \times \mathtt{20042553305308}$
- $\mathtt{P(12)} = 2^{-72} \times \mathtt{623729532807}$
- $\mathtt{P(13)} = 2^{-72} \times \mathtt{14354889437}$
- $\mathtt{P(14)} = 2^{-72} \times \mathtt{244322621}$
- $\mathtt{P(15)} = 2^{-72} \times \mathtt{3075302}$
- $\mathtt{P(16)} = 2^{-72} \times \mathtt{28626}$
- $\mathtt{P(17)} = 2^{-72} \times \mathtt{197}$
- $\mathtt{P(18)} = 2^{-72} \times \mathtt{1}$

One can check that for any $a \leq 513$, $R_a(P, D_{\mathbb{Z}^+,\sigma_0}) \leq 1 + 2^{-80} \leq 1 + \frac{1}{4Q_{\mathrm{bs}}}$, which in turn allows to apply Lemma 4.

Towards a faster option One possible way to improve the efficiency of the previous CDT technique would be to apply the recent work of Karmakar et al. [KRVV19] to obtain a Knuth-Yao tree from our CDT. Our work implies that one can apply [KRVV19] with more aggressive parameters (precisely, $\sigma_0 = 1.8205$ and $\theta = 72$, instead of $\sigma_0 = 2$ and $\theta = 128$ in [KRVV19]). As bit generation takes about 80% of the running time of the sampler of [KRVV19], this should lead to a much greater efficiency.

5.3 Choices for the implementation

We provide an implementation using floating-point arithmetic. It uses P_{facct} for computing the exp() and $\mathsf{BaseSampler} = \mathsf{BaseSampler}_{19,72}$.

We are currently working on improving this implementation from two angles. First, we expect to improve the performance with the use of [KRVV19] for our BaseSampler. Secondly, we expect to provide a *portable version with fixed precision arithmetics* by removing all floating points and specialized commands. For this, we plan to use an adapted polynomial for the exp with [BBE+19] and to change the computation of the Falcon tree to store the standard deviations and centers with fixed precision.

6 Analysis of resistance against timing attacks

In this section, we show that Algorithm 2 is impervious against timing attacks. We formally prove that it is constant-time in the sense of Definition 2, with the sensitive values being the inputs σ, μ and the output z.

Attack model. Let $\mu, \sigma \in \mathbb{R}$ be fixed, with $\sigma > \eta_\epsilon(\mathbb{Z})$. We suppose that an adversary $\mathcal{A}$ can monitor precisely the execution time of at most Q_{samplZ} executions of $\mathsf{SamplerZ}(\sigma, \mu)$. However, $\mathcal{A}$ does not know σ, μ, nor does he know the value z output at each execution. The goal of $\mathcal{A}$ is:

A. To estimate σ;
B. To estimate μ;
C. For each execution, to recover the integer z output.

These are the sensitive values for SamplerZ. During the execution of SamplerZ, three elements may leak the sensitive values:

1. The running time of BaseSampler;
2. The running time of BerExp;
3. The number of iterations of the **while** loop.

On the other hand, if BaseSampler, BerExp and the number of iterations of the **while** loop are constant-time, then SamplerZ is constant-time as a composition of constant-time elements (since the other steps are a composition of elementary arithmetic operations, which are considered constant-time). To prove that SamplerZ is constant-time, we prove that each of these elements is constant-time with respect to each sensitive value σ, μ, z. This requires to verify $3 \times 3 = 9$ conditions, organized in Table 2.

Table 2. Matrix of the conditions required to assess that SamplerZ is constant-time.

	σ	μ	z
BaseSampler	1A	1B	1C
BerExp	2A	2B	2C
# iterations	3A	3B	3C

6.1 Analysis of BaseSampler

We first show that BaseSampler is constant-time.

With respect to σ and μ. BaseSampler takes neither σ nor μ as inputs, so its running time is independent of them.

With respect to z. The running time of BaseSampler is a fixed constant (see Section 5.2), so it is assumed constant time.

6.2 Analysis of BerExp

We now show that BerExp is constant-time with respect to σ, μ and z. In BerExp, all the atomic operations are done in constant time, including the generation of u. Indeed, the polynomial approximation of Section 5.1 is constant time. The number of iterations of the **while** loop is independent of any secret value: at each iteration, the probability that the loop stops is 2^{-8}. Thus the running time of BerExp is independent of its input x and output b (thus of σ, μ and z).

6.3 Number of iterations of the while loop

In this section, we prove that the rejection rate (or equivalently, the number of iterations of the **while** loop) leaks no sensitive value (in our case μ, σ, z).

Choice of C. We now make explicit our choice of C in BerExp. For each σ, we take $C(\sigma) = \frac{\sigma_{\min}}{\sigma}$, where $\sigma_{\min} = \eta_\epsilon(\mathbb{Z}^n)$ is a lower bound on the possibles values of σ. Clearly, $C(\sigma) \in (0, 1)$ for the considered values of σ. This scaling factor is independent of μ, z, and we will see that it also makes the number of iterations independent of σ.

To simplify our argument, we suppose that BaseSampler and BerExp are perfectly correct; leveraging the same arguments as in Section 5, one can show that our argument hold as well when the real algorithms BaseSampler and BerExp are used. For fixed inputs σ and μ, the probability that a given $z \in \mathbb{Z}$ (with a uniquely associated z_0) is output by SamplerZ at an iteration is:

$$\mathbb{P}[\mathsf{SamplerZ} \to z] = \underbrace{\frac{\rho_{\sigma_0}(z_0)}{\rho_{\sigma_0}(\mathbb{Z}^+)}}_{\mathbb{P}[\mathsf{BaseSampler} \to z_0]} \cdot \underbrace{\frac{1}{2}}_{\mathbb{P}[b]} \cdot \underbrace{C(\sigma) \cdot \frac{\rho_{\sigma,\mu}(z)}{\rho_{\sigma_0}(z_0)}}_{\mathbb{P}[\mathsf{BerExp} \to \mathbf{true} | z]} = \frac{C(\sigma) \cdot \rho_{\sigma,\mu}(z)}{2 \cdot \rho_{\sigma_0}(\mathbb{Z}^+)}.$$

Let $P_{\mathbf{true}}$ denote the probability over an iteration that BerExp $\to$ **true**. We have:

$$P_{\mathbf{true}} = \mathbb{P}[\mathsf{BerExp} \to \mathbf{true}] = \sum_z \mathbb{P}[\mathsf{SamplerZ} \to z] = \frac{C(\sigma) \cdot \rho_{\sigma,\mu}(\mathbb{Z})}{2 \cdot \rho_{\sigma_0}(\mathbb{Z}^+)}, \tag{7}$$

which is proportional to $C(\sigma) \cdot \rho_{\sigma,\mu}(\mathbb{Z})$, since σ_0 is a public constant. For the values of σ_0, σ used in Falcon, it holds that $P_{\mathbf{true}} \leq 1/2$. Incidentally, this also implies that the average number of iterations $\mathcal{N}_{\text{iter}}$verifies $\mathcal{N}_{\text{iter}} \leq 2$. We now show that the distribution of the number of **while** loops in SamplerZ is constant-time, which is equivalent to showing that $P_{\mathbf{true}}$ is independent of μ, σ and z.

With respect to μ. Discarding factors independent of μ, the probability $P_{\mathbf{true}}$ is proportional to $\rho_{\sigma,\mu}(\mathbb{Z})$, see (7). In Falcon, by construction all the σ's verify $\sigma \geq \eta_\epsilon(\mathbb{Z}^n) \geq \eta_{\epsilon/n}(\mathbb{Z})$, with:

$$\epsilon \leq \frac{1}{\sqrt{4\lambda Q_s}}\ . \tag{8}$$

From [GPV08, Lemma 2.7], it holds that if σ verifies $\sigma \geq \eta_\delta(\mathbb{Z})$ for some $\delta \in (0, 1)$, then:

$$\rho_{\sigma,\mu}(\mathbb{Z}) \in \left[\frac{1-\delta}{1+\delta}, 1\right] \cdot \rho_\sigma(\mathbb{Z}). \tag{9}$$

We note $\mathcal{P} = \mathcal{B}_{P_{\mathbf{true}}}$ and $\mathcal{Q} = \mathcal{B}_{P_{\mathbf{true}} \cdot \frac{\rho_\sigma(\mathbb{Z})}{\rho_{\sigma,\mu}(\mathbb{Z})}}$. The distribution $\mathcal{Q}$ is ideal and is independent of μ, but the one we get in practice is $\mathcal{P}$. Noting $\delta = \epsilon/n$ and combining (9) with [Pre17] Lemma 3, the Rényi divergence $R_a(\mathcal{P}, \mathcal{Q})$ verifies:

$$R_a(\mathcal{P}, \mathcal{Q}) \leq \left(1 + \frac{a(a-1)\delta^2}{2(1-\delta)^{a+1}}\right)^{\frac{1}{a-1}} \underset{\delta \to 0}{\sim} 1 + \frac{a\delta^2}{2}, \tag{10}$$

Combining (10), (9), taking $a = 2\lambda + 1$ and following the same arguments as in Section 5, one can argue that recovering μ given the number of **while** loops is as hard as without this information, as long as there are $O(n^2 Q_s)$ calls to $\mathcal{P}$. In comparison, the actual number of calls to $\mathcal{P}$ is smaller by a factor $O(n)$, see Table 1.

With respect to σ. This is where our scaling function plays an important role. We have seen that BerExp outputs **true** with a probability essentially proportional to $C(\sigma) \cdot \rho_\sigma(\mathbb{Z})$. By the Poisson summation formula:

$$\rho_\sigma(\mathbb{Z}) = \sigma\sqrt{2\pi} \cdot \left(1 + 2\sum_{n \geq 1} \exp(-2n^2\pi^2\sigma^2)\right) \approx \sigma\sqrt{2\pi} \tag{11}$$

Thus the probability that BerExp outputs **true** is essentially proportional to $C(\sigma) \cdot \sigma = \sigma_{\min}$.[8] Therefore, the number of iterations of the **while** loop is independent of σ.

Impact of the scaling factor. We study the impact of the scaling factor $C(\sigma)$ on the running time. In Falcon, 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:

$$\max_i \|\tilde{\mathbf{b}_i}\| \leq 1.17\sqrt{q} \tag{12}$$

$$\min_i \|\tilde{\mathbf{b}_i}\| \geq \sqrt{q}/1.17 \tag{13}$$

By construction, (12) is true (Falcon enforces this condition). To prove (13), we rely on a peculiar property of Falcon's private bases: *symplecticity*. Let $\mathbf{J} = \left[\begin{array}{c|c} 0 & 1 \\ \hline -1 & 0 \end{array}\right]$, and let $\mathbf{B} = \left[\begin{array}{c|c} g & -f \\ \hline G & -F \end{array}\right]$ be a private basis used in Falcon. It has been observed in [GHN06] that $\mathbf{B}$ is q-symplectic, that is, it verifies:

$$\mathbf{B}^t \times \mathbf{J} \times \mathbf{B} = q \cdot \mathbf{J}. \tag{14}$$

As per [GHN06, Corollary 1], this implies that for any i, $\|\tilde{\mathbf{b}}_{2n+1-i}\| = q/\|\tilde{\mathbf{b}}_i\|$. Combining this with (12) yields (13). Thus $C(\sigma) \leq \frac{\sigma_{\min}}{\sigma_{\max}} \leq (1.17)^{-2} \approx 0.73$, which means a non-negligible but reasonable impact on the running time of the sampler.

With respect to z. It is immediate from (7) that $P_{\mathbf{true}}$ is independent of z. Thus the output z is *independent* of the number of iterations of the **while** loop.

7 Benchmarks

In this section, we first discuss the results of our constant-time implementation of Falcon. Next, we compare it with the reference implementation running times of Falcon and Dilithium.

7.1 Results

In our constant-time implementation of Falcon, the cost of the signing operation is dominated by the cost of the fast Fourier sampler as this component accounts for 76% of its total cycle count. In turn, the cost of the fast Fourier sampler heavily depends on the performance of the Gaussian sampler that is executed $2n$ times during the fast Fourier sampling. The $2n$ calls to the Gaussian sampler account for 58% of the cycle count of the entire signature generation. Finally, the cost of the fast Gaussian sampler heavily depends on the performance of the PRNG (a portable implementation of ChaCha20) whose cost accounts for 28% of the entire signature generation.

[8] The approximation in (11) is precise enough to make this simplification. Indeed, for the parameters of Falcon, it has a relative error $\leq 2^{-44}$, at which point one can apply the same Rényi arguments as previously.

7.2 Comparison

In Table 3 we present the running times of our constant-time implementation on an Intel Core i7-6500U CPU (clocked at 2.5 GHz) and compare them with the reference implementation running times of Falcon and Dilithium.

- The Falcon reference implementation is not constant-time. It uses only standard C code, no inline assembly, intrinsic or 128-bit integer. It uses a standard implementation of ChaCha20 as PRNG.
- The Dilithium reference implementation is constant-time, it does not branch depending on secret data and does not access memory locations that depend on secret data. The reference implementation we consider is the non-AVX2 and non-AES one. It uses SHAKE-256 and SHAKE-128 as PRNG.
- Our constant-time Falcon implementation uses only standard C code, no inline assembly, floating-point division, intrinsic or 128-bit integer, except to compute the exp function by using XMM registers of 128 bit [ZSS18,KRVV19]. The PRNG is unchanged compared to the reference implementation of Falcon.

Table 3. Comparison of our constant-time implementation of Falcon with the reference implementation of Falcon and the constant-time reference implementation of Dilithium.

NIST Level	Scheme	Constant-Time	sign/s	vrfy/s	pub length	sig length
1 - AES128	Falcon (reference)	No	5905	41152	897	618
	Falcon (this work)	Yes	4027	37952	897	618
	Dilithium (reference)	Yes	2062	10670	1184	2044
2 - SHA256	Dilithium (reference)	Yes	1343	7310	1472	2701
3 - AES192	Dilithium (reference)	Yes	1507	5578	1760	3366
5 - AES256	Falcon (reference)	No	3093	20085	1793	1234
	Falcon (this work)	Yes	2097	20260	1793	1234

Signature generation time does not include the Falcon LDL tree building, which is done when the private key is loaded. Compared to the reference implementation, ours is about 50% slower, but remains very competitive speed-wise. In addition, one can expect a much closer gap using intrinsics (such as AES-NI) for the PRNG and replacing the CDT by Knuth-Yao trees as per [KRVV19].

Acknowledgements

We thank Léo Ducas for helpful suggestions. We also thank Thomas Pornin and Mehdi Tibouchi for useful discussions. The first author was supported by the project PQ Cybersecurity (Innovate UK research grant 104423). The second and third 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 third author was also supported by ANRT under the program CIFRE N2016/1583.

References

AD73. Joachim Ahrens and Ulrich Dieter. Extension of forsythe's method for random sampling from the normal distribution. *Mathematics of computation*, 27:927–937, 1973. 2

BBE+19. 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. https://eprint.iacr.org/2019/511. 1, 2, 9, 12

BDE+18. Jonathan Bootle, Claire Delaplace, Thomas Espitau, Pierre-Alain Fouque, and Mehdi Tibouchi. LWE without modular reduction and improved side-channel attacks against BLISS. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part I*, volume 11272 of *LNCS*, pages 494–524, Brisbane, Queensland, Australia, December 2–6, 2018. Springer, Heidelberg, Germany. 1

BLL+15. 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, Auckland, New Zealand, November 30 – December 3, 2015. Springer, Heidelberg, Germany. 2, 3

BPSV19. Ward Beullens, Bart Preneel, Alan Szepieniec, and Frederik Vercauteren. LUOV. Technical report, National Institute of Standards and Technology, 2019. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions. 1

DDLL13. 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, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany. 1, 2

DG14. 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, Jun 2014. 2

DN12. 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, Beijing, China, December 2–6, 2012. Springer, Heidelberg, Germany. 2

Duc13. Léo' Ducas. *Signatures fondées sur les réseaux euclidiens : attaques, analyses et optimisations*. Theses, École Normale Supérieure, 2013. 2

DWZ18. 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. 2

EFGT17. 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. [TEMX17], pages 1857–1874. 1

Est60. 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. 10

For72. 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. 2

GHN06. 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, St. Petersburg, Russia, May 28 – June 1, 2006. Springer, Heidelberg, Germany. 14

GPV08. 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, Victoria, BC, Canada, May 17–20, 2008. ACM Press. 13

HBD+19. Andreas Hulsing, Daniel J. Bernstein, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Panos Kampanakis, Stefan Kolbl, Tanja Lange, Martin M Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, Peter Schwabe, and Jean-Philippe Aumasson. SPHINCS+. Technical report, National Institute of Standards and Technology, 2019. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions. 1

HLS18. 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, Rio de Janeiro, Brazil, March 25–29, 2018. Springer, Heidelberg, Germany. 2

Kar16. Charles F. F. Karney. Sampling exactly from the normal distribution. *ACM Trans. Math. Softw.*, 42(1):3:1–3:14, January 2016. 2

KRVV19. Angshuman Karmakar, Sujoy Sinha Roy, Frederik Vercauteren, and Ingrid Verbauwhede. Pushing the speed limit of constant-time discrete gaussian sampling. a case study on falcon. Cryptology ePrint Archive, Report 2019/267, 2019. https://eprint.iacr.org/2019/267. 1, 2, 11, 12, 15

LDK+19. 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. 1, 2

MR07. Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on gaussian measures. *SIAM J. Comput.*, 2007. 3

MW17. 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, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany. 2

NIS16. NIST. Submission requirements and evaluation criteria for the post-quantum cryptography standardization process, 2016. https://csrc.nist.gov/Projects/Post-Quantum-Cryptography. 4

PBY17. 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. [TEMX17], pages 1843–1855. 1

PDG14. 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, Busan, South Korea, September 23–26, 2014. Springer, Heidelberg, Germany. 2

PFH+17. 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, 2017. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions. 1

PFH+19. 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. 4

Pre17. 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, Hong Kong, China, December 3–7, 2017. Springer, Heidelberg, Germany. 2, 3, 13

TEMX17. Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors. *ACM CCS 2017*, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press. 16, 17

von50. John von Neumann. Various techniques used in connection with random digits. *National Bureau of Standards, Applied Math Series*, 12:36–38, 11 1950. 2

Wal19. Michael Walter. Sampling the integers with low relative error. Cryptology ePrint Archive, Report 2019/068, 2019. https://eprint.iacr.org/2019/068. 2

ZSS18. Raymond K. Zhao, Ron Steinfeld, and Amin Sakzad. Facct: Fast, compact, and constant-time discrete gaussian sampler over integers. Cryptology ePrint Archive, Report 2018/1234, 2018. https://eprint.iacr.org/2018/1234. 1, 2, 9, 15