Addressing Side-Channel Vulnerabilities in the Discrete Ziggurat Sampler

Séamus Brannigan, Máire O'Neill, Ayesha Khalid, and Ciara Rafferty

Centre for Secure Information Technologies (CSIT), Queen's University Belfast, UK
`sbrannigan11@qub.ac.uk`

Abstract. Post-quantum cryptography with lattices typically requires high precision sampling of vectors with discrete Gaussian distributions. Lattice signatures require large values of the standard deviation parameter, which poses difficult problems in finding a suitable trade-off between throughput performance and memory resources on constrained devices. In this paper, we propose modifications to the Ziggurat method, known to be advantageous with respect to these issues, but problematic due to its inherent rejection-based timing profile. We improve upon information leakage through timing channels significantly and require: only 64-bit unsigned integers, no floating-point arithmetic, no division and no external libraries. Also proposed is a constant-time Gaussian function, possessing all aforementioned advantageous properties. The measures taken to secure the sampler completely close side-channel vulnerabilities through direct timing of operations and these have no negative implications on its applicability to lattice-based signatures. We demonstrate the improved method with a 128-bit reference implementation, showing that we retain the sampler's efficiency and decrease memory consumption by a factor of 100. We show that this amounts to memory savings by a factor of almost $5,000$, in comparison to an optimised, state-of-the-art implementation of another popular sampling method, based on cumulative distribution tables.

1 Introduction

Lattice-based Cryptography (LBC) has become popular in the field of post-quantum public-key primitives and aids research into more advanced cryptographic schemes such as fully-homomorphic, identity-based and attribute-based encryption. For a thorough review on applications and background of LBC, see [1]. This attention is partly due to the low precision arithmetic required to implement a lattice scheme, which rarely extends beyond common standard machine word lengths. The algorithmic complexities are based on vector operations over the integers.

There is one, increasingly contentious, component which requires extra precision: Gaussian sampling. By cryptographic standards, this extra precision is low and begins and ends in the sampling phase. First introduced theoretically to LBC in [2], Gaussian sampling has been shown to reduce the required key sizes of lattice schemes, but also to be prone to side channel attacks. As an example of this, an attack [3] on the sampler in the lattice-based signature scheme, BLISS [4], has been demonstrated using timing differences due to cache misses.

Regardless of the push toward other solutions for cryptographic primitives, Gaussian sampling is prevalent in LBC. It appears in the proofs of security of the fundamental problems [2] and the more advanced applications, especially those using lattice trapdoors [5], rely on it. Each of these applications will be expected to adapt to constrained devices in an increasingly connected world. The NIST call for post-quantum cryptographic standards [6] has resulted in a large number of lattice-based schemes being submitted, of which a significant proportion use Gaussian sampling [7,8,9].

Issues around the timing side channel exposed by the Gaussian sampling phase would ideally be dealt with by implementing outright constant-time sampling routines. However, popular candidates for LBC include the CDT [10] and Knuth/Yao [11] samplers, based on cumulative distribution tables and random tree traversals, respectively. The impact of ensuring constant-time sampling with these methods is a reduction in their performance.

1.1 Related Work

The large inherent memory growth of these samplers with increasing precision and standard deviation, combined with constant-time constraints, prompted the work of Micciancio and Walter [12]. An arbitrary base sampler was used to sample with low standard deviation, keeping the memory and time profile low, then convolutions on the Gaussian random variables were used to produce samples from a Gaussian distribution with higher standard deviation. The result was a significant reduction in the memory required to sample the same distribution with just the base sampler, with no additional performance cost. Importantly, given a constant-time base sampler operating at smaller standard deviation, the aggregate method for large standard deviation is constant-time.

The Micciancio-Walter paper boasts a time-memory trade off similar to that of Buchmann et al.'s Ziggurat sampler [13]. The former outperforms the latter as an efficient sampler, but the latter has a memory profile better suited to constrained devices. It can be seen in the results of [12] that the convolution method's lowest memory usage is at a point where the Ziggurat has already maximised its increasing performance. The potential performance of the Ziggurat method exceeds that of the CDT, for high sigma, the latter being commonly used as a benchmark. We ported the former to ANSI C using only native 64-bit `double` types, we compared their performances and memory profiles, finding the Ziggurat to be favourable for time and space efficiency, for increasing size of inputs and parameters. See Figure 1 for throughput performance and Table 1 for memory consumption. This comparison is the first of its kind, where Buchmann's Ziggurat has been implemented in ANSI C, free from the overhead of the NTL library and higher-level C++ constructs, as the CDT and others have been.

The problem with the Ziggurat method is that it is not easy to contain timing leakage from rejection sampling. The alternative is to calculate the exponential

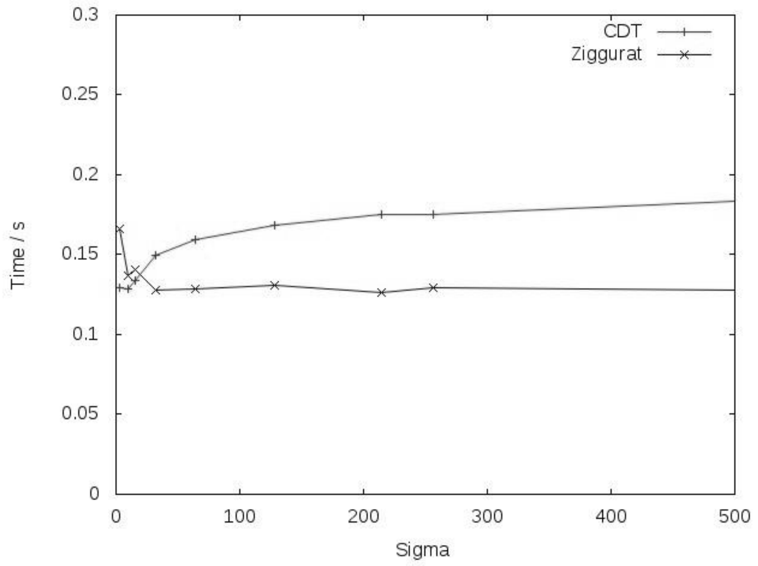

Fig. 1: Time taken for preliminary Ziggurat and CDT samplers to sample 1 million Gaussian numbers. These early experiments were done to 64 bit precision using floating point arithmetic on one processor of an Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz

function every time. But it is the exponential function, in fact, which causes the most difficulty. Both the NTL [14], used in [13], and glibc [15], used in Figure 1, exponential functions are prone to leakage, the former from early exit of a Taylor series and the latter from proximity to a table lookup value.

Sampler	Memory usage(Bytes)
CDT	32,778
Ziggurat	1,068

Table 1: Memory usage of the 64-bit CDT and Ziggurat samplers at $\sigma = 215$. Value for Ziggurat is for 64 rectangles, where its performance peaks.

1.2 Our Contribution

We build on the work of Buchmann et al. [13] by securing the Ziggurat sampler with respect to information leakage through the timing side channel. The algorithms proposed in this paper target schemes which use a large standard deviation on constrained devices.

- We highlight side-channel vulnerabilities in the Ziggurat method, not mentioned in the literature, and propose solutions for their mitigation.
- The Ziggurat algorithm is redesigned to prevent leakage of information through the timing of operations.
- We propose a novel algorithm for evaluating the Gaussian function in constant time. To the best of our knowledge, this is the first such constant-time algorithm.
- The Gaussian function, and the overall Ziggurat sampler, is a fixed-point algorithm built from 64-bit integers, using no division or floating point arithmetic, written in ANSI C.
- The reference implementation achieves similar performance to the original sampler by Buchmann et al. and, as it is optimised for functionality over efficiency, we expect the performance can be further improved upon.
- The amount of memory saved by using our algorithm is significantly greater than the advantage seen, already, in the original sampler.
- We argue that the proposed sampler now has sufficient resilience to physical timing attacks to be considered for constrained devices (such as microcontrollers) and hardware implementations not making use of a caching system.

The paper is organised as follows. After a preliminary discussion in Section 2, Gaussian sampling via the Ziggurat method of [13] is outlined in Section 3. The new fixed-point Ziggurat algorithm is described in Section 4, as is the new fixed-point, constant-time Gaussian function, in Section 4.2. We discuss the results of the sampler and the security surrounding the timing of operations in Section 5.

2 Preliminaries

Notation We use the shorthand $\{x_i\}_{i=a}^{n} \overset{def}{=} \{x_i | i \in \mathbb{Z}, a \leq i \leq n\}$. When dealing with fixed-point representations of a number x, we refer to the fractional part as $x_{\mathbb{Q}}$ and the integer part as $x_{\mathbb{Z}}$. The same treatment is given to the results of expressions of mixed numbers, where the expression is enclosed in parentheses and subscripted accordingly. The approximate representation of a number y is denoted $\bar{y}$.

Discrete Gaussian Sampling A *discrete Gaussian distribution* $\mathcal{D}_{\mathbb{Z},\sigma}$ over $\mathbb{Z}$, having 0 mean and a standard deviation denoted by σ, is defined as $\rho_\sigma(x) = \exp(-x^2/2\sigma^2)$ for all integers $x \in \mathbb{Z}$. the support, β, of $\mathcal{D}_{\mathbb{Z},\sigma}$ is the (possibly infinite) set of all x which can be sampled from it. The support can be superscripted with a $+$ or $-$ to indicate only the positive or negatives subsets of β and a zero subscripted to either of these to indicate the inclusion of 0.

Considering $S_\sigma = \rho_\sigma(\mathbb{Z}) = \sum_{k=-\infty}^{\infty} \rho_\sigma(k) \approx \sqrt{2\pi}\sigma$, the sampling probability for $x \in \mathbb{Z}$ from the Gaussian distribution $\mathcal{D}_{\mathbb{Z},\sigma}$ is calculated as $\rho_\sigma(x)/S_\sigma$. For the LBC constructions undertaken in this research, σ is assumed to be fixed and known, hence it suffices to sample from $\mathbb{Z}^+$ proportional to $\rho(x)$ for all $x > 0$ and to set $\rho(0)/2$ for $x = 0$, where a sign bit is uniformly sampled to output values over $\mathbb{Z}$.

Other than the standard deviation, σ, and the mean, $c = 0$ for brevity, there are two critical parameters used to describe a finitely computed discrete Gaussian distribution. The first is the precision parameter, λ, which governs the statistical distance between the finitely represented probabilities of the sampled distribution and the theoretical Gaussian distribution with probabilities in $\mathbb{R}^+$.

The second is the tail-cut parameter, τ, which defines how much of the Gaussian distribution's infinite tail can be truncated, for practical considerations. This factor multiplies the σ parameter to give the maximum value which can be sampled, such that $\beta = \{x \mid 0 \leq x \leq \lceil \tau\sigma \rceil\}$. The choice of λ and τ affects the security of LBC schemes, the proofs of which are often based on the theoretical Gaussian distribution. The schemes come with recommendations for these, for a given security level.

The parameters λ and τ are not independent of each other. Sampling to λ-bit precision corresponds to sampling from a distribution whose probabilities differ by, at most, $2^{-\lambda}$. The tail is usually cut off so as not to include those elements with combined probability mass below $2^{-\lambda}$. By the definition of the Gaussian function, this element occurs at the same factor, τ, of σ. For 128-bit precision, $\tau = 13$, and for 64-bit precision, $\tau = 9.2$.

Taylor Series Approximation The exponential function, e^x, expands as a Taylor series evaluated at zero like such, $e^x = \sum_{i=0}^{\infty} x^i/i!$. When the term to be summed is $< 2^{-\lambda}$, the function has been approximated to λ bits.

3 Discrete Ziggurat Sampling

The Ziggurat sampling technique of Marsaglia and Tsang [16], is a rectangle-wedge approach to rejection sampling originally proposed for both normal and exponential distributions. The basic method of 'rejection' sampling a distribution is to uniformly sample two numbers, x and y. If $y \leq \rho_\sigma(x)$, then x is returned as the sampled variable. In other words, x is rejected with probability determined by the distribution $\mathcal{D}_\sigma$. The computational expense in calculating $\rho_\sigma(x)$, or, the alternative, storing all the information in the distribution, motivated the development of the Ziggurat method.

The distribution is enclosed by a set of m rectangles, $\{\mathcal{R}_i\}_{i=1}^n$, such that the bottom-right corner of each rectangle is a point on the distribution. Figure 1 shows the first few and the m^{th} rectangles of such a set. Each $\mathcal{R}_i$, in the continuous case, is the area given by $x_i(y_{i-1} - y_i)$ and R_0 is simply the x_0 co-ordinate. A continuous description is given here and then adapted for the discrete case.

For each $\mathcal{R}_{i\neq 1}$, there is a continuous set of $x_j \leq x_{i-1}$ such that every y_j within $\mathcal{R}_i$ is completely under the distribution and such that every rectangle contains the same 2-dimensional sample space. In the continuous case, this corresponds to rectangles of equal area. It is therefore possible to uniformly sample an $\mathcal{R}_i$ and accept the majority of x_js without having to compute $f(x) = \rho_\sigma(x)$. Increasing the number of rectangles covering the distribution decreases the probability of x being in the 'rejection zone' and improves the efficiency of the algorithm.

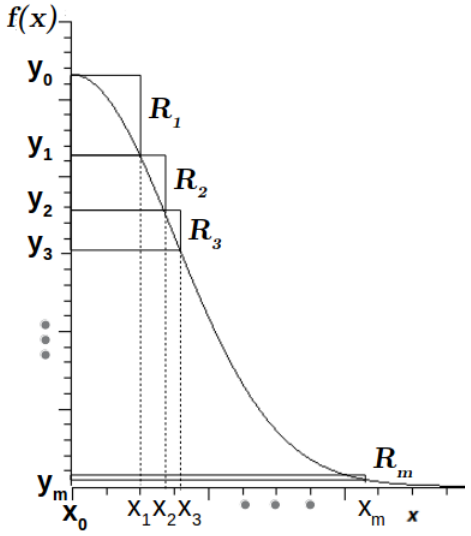

Fig. 2: The Ziggurat setup. Rectangles of equal area enclose a Gaussian distribution.

However, more rectangles require more storage and a balance must be found between time and space. With the Ziggurat method, $f(x)$ need only be computed in a relatively low number of instances; when the number to be sampled is in the rejection zone. Also, the second co-ordinate, y, need only be generated in these instances. Generating a random integer within the range of the number of rectangles is more efficient than generating a number to the required precision of the sampled distribution.

The discrete Gaussian distribution, and the Ziggurat method for sampling from it, are similar to their continuous counterparts. The intuitive difference is that the distribution now takes the form of a histogram. The x value which multiplies the y value to give the common area, as opposed to being the horizontal distance from zero to x_i, is now the number of integer values on the x- axis contained within the rectangle. This will be the value $\lfloor x \rfloor + 1$, to account for zero.

The discrete Ziggurat of [13] is summarised in Algorithm 1, although note that we have omitted the sLine() phase, for simplicity, as we do not use it in our algorithm. For now, we wish to simplify and merge the paths through the sampler, so we restrict the algorithm to its minimally essential form.

In Algorithm 1, the 2-dimenional sample space is uniformly sampled in y, i.e. an $\mathcal{R}_i$ is chosen, then uniformly sampled in x over $\{\lfloor x_i \rfloor\}_{i=1}^{m}$. These samples are mostly accepted. In some cases, when x is higher than x_{i-1}, sampling to finer precision is needed in y. Then, the vertical space of the rectangle is discretised into 2^λ elements and uniformly sampled. Should this point lie within the vertical

Algorithm 1 Zig_Ntl_Sample(m, σ, λ, $\{\lfloor x_i \rfloor\}_{i=1}^{m}$, $\{\bar{y}_i\}_{i=0}^{m}$)

```
 1: while true do
 2:     $r \xleftarrow{\$} \{i\}_{i=1}^{m}, s \xleftarrow{\$} \{-1, 1\}, x \xleftarrow{\$} \{i\}_{i=0}^{x_r}$
 3:     if $0 < x \leq x_{i-1}$ then
 4:         return sx
 5:     else
 6:         if $x = 0$ then
 7:             $b \xleftarrow{\$} \{0, 1\}$
 8:             if $b = 0$ then
 9:                 return $sx$
10:             else
11:                 continue
12:         else
13:             $y' \xleftarrow{\$} \{i\}_{i=0}^{2^\lambda - 1}$, $\bar{y} = y' \cdot (\bar{y}_{i-1} - \bar{y}_i)$
14:             if $\bar{y} \leq \bar{\rho}_\sigma(x) - \bar{y}_i$ then
15:                 return $sx$
16:             else
17:                 continue
```

region covered by the distribution, it is accepted. Rejection of a sample causes the process to begin again, until the function outputs a sample.

4 Fixed-Point Ziggurat Method with Time Considerations

Here, we describe and analyse how the proposed sampler operates. Specifically, this section details the novel contribution of this work. Section 4.1 provides an overview of how the Ziggurat method has been adapted to consider the timing side-channel and suitability for constrained devices at parameters for lattice signatures, namely, high standard deviation, σ. Section 4.2 sets up the theoretical basis of the constant-time Gaussian function with Theorem 1 and discusses the required input precision, with Theorem 2. Finally, the Gaussian function is given explicitly in Algorithm 4 and described throughout the section.

4.1 Timing Attack Resilient, Time-Independent Ziggurat Sampling

A purely constant-time rejection sampler over discrete Gaussians is hard to envisage, apart from one that calculates the probability function at every sample, which is a low memory, low performance extremum. Rather than focus on sample by sample uniformity in the temporal distribution of the sampler, we reduce the number of possible timings of the Ziggurat method from arbitrarily many, depending on how the Gaussian function is called, to two: sampling in the rejection zone in constant time over the integers, or accepting straight away in constant time over the integers.

Apart from the constant-time routine for the Gaussian function, there are a few paths through the Ziggurat which need to be merged before the above can be done. As can be seen in Algorithm 1, the original Ziggurat method will take a unique path when $x = 0$. An attacker with the ability to time the operations of the sampler would, hence, know those samples with value zero, which are also the most frequent. Not all paths through the algorithm are as obviously insecure. For instance, should $x = 0$ be rejected, the attacker still gains information about the state of the underlying PRNG. Either way, information leakage of the kind which gives an attacker a high degree of confidence in the values of variables in the sampler (near certainty, in this case), is required to be mitigated.

Algorithm 2 shows the proposed Ziggurat sampling algorithm. Note that Algorithm 3 is the function which calls $\rho_\sigma(x)$, the Gaussian function of Algorithm 4. For descriptions of the other functions called, see the prose which follows.

Algorithm 2 ZIGGURAT_SAMPLE$(m,\ s,\ \lambda,\ \{\lfloor x_i \rfloor\}_{i=1}^{m},\ \{\bar{y}_i\}_{i=0}^{m})$

```
 1: while true do
 2:     $r \xleftarrow{\$} \{i\}_{i=1}^{m}, s \xleftarrow{\$} \{0,1\}, x \xleftarrow{\$} \{i\}_{i=0}^{2^\lambda - 1}$
 3:     $x = \lfloor x \cdot (x_r + 1) \rfloor$                                    ▷ $x/2^\lambda \mapsto \mathcal{R}_r$
 4:     acc = ct_lte$(x, x_{r-1}) \wedge ($ct_isnonzero$(x) \vee s)$
 5:     if acc then
 6:         break
 7:     else
 8:         acc = Ziggurat_Sample_y$(x, y_{r-1}, y_r, \sigma, \lambda)$
 9:         acc = acc $\wedge$ ct_isnonzero$(x)$
10:         if acc then                                                   ▷ $y \leq y'$
11:             break
12: return $x -$ ct_select$(0, 2x, s)$
```

Algorithm 3 ZIG_SAMPLE_Y$(x, \bar{y}_b, \bar{y}_a, \sigma, \lambda)$

```
1: $y' \xleftarrow{\$} \{i\}_{i=0}^{2^\lambda - 1}$
2: $\bar{y} = y' \cdot (\bar{y}_b - \bar{y}_a)$
3: return ct_lte$(\bar{y},\ \rho_\sigma(x) - \bar{y}_a)$
```

In Algorithm 2, the table of y values of the rectangles, the $\{\bar{y}_i\}_{i=0}^{m}$, are p-bit unsigned integers representing numbers in $[0, 1)$. For all cryptographic purposes, p is greater than the length of machine words and requires high precision arithmetic. $\{\lfloor x_i \rfloor\}_{i=1}^{m}$ are unsigned integers which normally fit within 32 bits. Only when σ is a value higher than those which have so far been proposed, does this change.

Uniform sampling of $\{x_i\}$ to p-bit precision is performed by sampling x to p bits from a cryptographically secure pseudo-random number generator

(CSPRNG). This number is interpreted as an integer representing the numerator of a fixed-point fraction in $[0, 1)$. Thus, by multiplying this uniformly random fraction by the discrete size of the rectangle, as in Line 3 of Algorithm 2, and taking the floor, we get a uniform sample in the rectangle.

The important novelty in this algorithm, with regards to timing, is the pair of accept (acc) conditions. In Algorithm 1, if a non-zero sample was accepted, it was negated with probability 1/2. If the sample was zero, it was accepted with probability 1/2. We use this fact in Algorithm 2 to handle these cases in the same computational step. Line 4 gives the logical shortcut to the desired outcome. Before describing this shortcut, a note on constant-time logical operations follows.

All functions beginning `ct_` are constant-time functions which return a 0 or a 1 as an unsigned integer. As an example, `ct_lte(a,b)` returns 1 if $a \leq b$ and 0 otherwise. All logical operations in these algorithms are implemented as bitwise operations on values returned from these functions. Hence, the logical binary operations can be synthesised in constant time by bitwise operations restricted to values of 0 and 1.

The particular logic of Line 4 comes from the fact that the same bit is used to determine if the case $x = 0$ is accepted, as is used to determine the sign of non-zero accepted samples. The logic for accepting is thus $(x = 0 \rightarrow s) \wedge x \leq x_{i-1}$. As $P \rightarrow Q \equiv \neg P \vee Q$, we get Line 4.

If the accept condition holds, the loop breaks and the sample is returned in Line 12. If it does not hold, the algorithm goes into the rejection phase. The algorithm sends rejected $x = 0$ samples through a redundant rejection phase, to prevent a timing attack revealing such a rejection. Thus, an attacker can know when a sample has been rejected, which is probably unavoidable with a rejection sampler, but not what the sample was. This is crucial for ensuring that the state of the underlying CSPRNG is not compromised.

The loop will continue until it breaks, in which case a sample will be ready to be returned. A constant-time select function, `ct_select(`a`, `b`, `c`)`, returns a if $c = 0$ and b if $c = 1$. Thus, Line 4 converts a sample $x \neq 0$ to a negative if the sign bit is set and leaves it alone if not. This operation will leave an $x = 0$ sample alone and the sampler will have two possible timings for an accepted sample and all rejections traverse the same computational path. If the function $\rho_\sigma(x)$ is made constant-time, the Ziggurat sampler is now significantly more robust against side-channel analysis.

4.2 Constant-Time Gaussian Evaluation

The Ziggurat sampler requires the evaluation of the exponential function to high precision, which must be done in constant time if it is to be suitable for cryptographic purposes. This is the fundamental design specification. The exponential function must also preserve, if not accentuate, many of the advantageous qualities of the Ziggurat sampler. Particularly, the Ziggurat method offers comparable performance to the CDT and Knuth/Yao samplers, but at a fraction of the memory consumption. This quality makes it a desirable candidate for hardware and embedded lattice-based cryptosystems.

Accordingly, the exponential function must have a small memory footprint, require as few hardware features (e.g. floating point arithmetic) as possible and avoid hardware-expensive division. The 8kB tables of glibc's standard 128-bit exp() function [15], for example, would triple the memory required for a 128-bit Ziggurat sampler with 128 rectangles. The lack of large lookup tables will result in a performance hit. However, there are numerous areas where at least some of this penalty can be diminished.

For example, the use of unsigned integers instead of floating point types and the replacement of divisions with multiplications should soften the penalty incurred. Combining this with the fact that the Ziggurat can be tuned so that calls to exp() will be made only for a small fraction of samples, the performance should remain comparable to that of the competing samplers.

Several challenges arise from the design criteria:

– Generating multi-precision arithmetic operations from the largest unsigned integer types which can be deemed standard (64 bits in this paper).
– Avoiding division for rational approximations, where division is a common component.
– Utilising these operations to mimic the floating point operations often used to approximate real numbers.
– Maintaining the Ziggurat's light-weight memory profile, whilst ensuring that the performance is comparable to other attempts at extending to high σ or λ.

Mathematical Underpinnings Recall that the Gaussian function is the evaluation of the exponential function over negative reals.

Theorem 1. *The evaluation of the exponential function $f(x') = \exp(x')$, $\forall x' \in \mathbb{R}_0^-$ and $f(x') \in [0, 1)$, can be formulated to output an integer in $\mathbb{Z}_q$, where $q = 2^\lambda$, representing the numerator of the closest fraction, over q, to $f(x')$. The problem is transformed to that of calculating a left shift,*

$$l_{\mathbb{Z}} = \left(\log_2 e \cdot (\lambda \cdot \ln 2 - s \cdot x^2)_{\mathbb{Z}}\right)_{\mathbb{Z}}, \tag{1}$$

and $y_\chi = \exp(\chi)$, for the fractional exponent

$$\chi = \ln 2 \cdot \left(\log_2 e \cdot (\lambda \cdot \ln 2 - s \cdot x^2)_{\mathbb{Z}}\right)_{\mathbb{Q}} + (\lambda \cdot \ln 2 - s \cdot x^2)_{\mathbb{Q}} \tag{2}$$

Proof. The objective is to calculate y' such that

$$\frac{y'}{2^\lambda} \approx y = e^{-x^2/2\sigma^2}, \tag{3}$$

for all $x \in \beta^+$. Changing the denominator of the left hand side to base e and rearranging gives

$$y' = e^{-x^2/2\sigma^2 + \lambda \ln 2}. \tag{4}$$

Let $x' = -x^2/2\sigma^2$ and $k = x' + \lambda \ln 2$, then observe that the range of values input to the exponential function shifts from $-\tau^2/2 \leq x' \leq 0$ to $\lambda \ln 2 - \tau^2/2 \leq k \leq \lambda \ln 2$. For $\lambda = 128$ and $\tau = 13$, for example, the range is from ~ 4.2 to ~ 88.7. Also, $y' \in \mathbb{Z}_{2^\lambda}$, always.

The new exponent k will consist of an integer part, $k_\mathbb{Z}$, and fractional part, $k_\mathbb{Q}$. Thus,

$$\begin{aligned} e^{k_\mathbb{Z} + k_\mathbb{Q}} &= e^{k_\mathbb{Z}} \cdot e^{k_\mathbb{Q}} & (5) \\ &= 2^{k_\mathbb{Z} \log_2 e} \cdot e^{k_\mathbb{Q}}, & (6) \end{aligned}$$

where a change to base 2 is made to convert the integer exponentiation to a shift on the result of the fractional exponentiation. Before this can be done, the fractional part of $k_\mathbb{Z} \log_2 e$ must be subtracted and added back into the fractional exponentiation.

Let $l = k_\mathbb{Z} \log_2 e$. Hence,

$$\begin{aligned} 2^l &= 2^{l_\mathbb{Z}} \cdot 2^{l_\mathbb{Q}} & (7) \\ &= 2^{l_\mathbb{Z}} \cdot e^{l_\mathbb{Q} \cdot \ln 2} & (8) \end{aligned}$$

and, therefore,

$$e^k = 2^{l_\mathbb{Z}} \cdot e^{l_\mathbb{Q} \cdot \ln 2 + k_\mathbb{Q}}. \tag{9}$$

Hence, the final left shift is $l_\mathbb{Z}$, and the input to the Gaussian Taylor Series is $\chi = l_\mathbb{Q} \cdot \ln 2 + k_\mathbb{Q}$. Here,

$$k = \lambda \cdot \ln 2 - s \cdot x^2 \tag{10}$$

and

$$l = k_\mathbb{Z} \cdot \log_2 e \tag{11}$$

■

Theorem 1 shows that the Gaussian function can be approximated with an integer, so long as a suitable approximation method is used for y_χ. Algorithm 4 presents the Gaussian function explicitly.

The design criteria which limits the choice of approximation method the most is the absence of division. For example, whereas methods such as continued fractions converge more rapidly, they require division by the input value. As the input values cannot be stored as precomputed fixed-point fractions, the criteria demands that the algorithm does not divide by the input. Hence, the only (immediately obvious) choice for the approximation method is the Taylor series. Theorem 1 is useful because, without converting the integer component of the exponentiation to a shift, the terms of the Taylor series, although converging, would contain x to too high a power to efficiently store and process.

Algorithm 4 $\rho(x, s, \lambda)$

```
1: Require: $\{f_i = \frac{1}{i!}\}_{i=1}^N$    ▷ $N$ s.t. $f_{N+1} < 2^{-\lambda}$
2: $k_{\mathbb{Z}} = (\lambda \cdot \ln 2 - s \cdot x^2)_{\mathbb{Z}}$
3: $k_{\mathbb{Q}} = (\lambda \cdot \ln 2 - s \cdot x^2)_{\mathbb{Q}}$
4: $l_{\mathbb{Z}} = \big(k_{\mathbb{Z}} \cdot \log_2 e\big)_{\mathbb{Z}}$
5: $l_{\mathbb{Q}} = \big(k_{\mathbb{Z}} \cdot \log_2 e\big)_{\mathbb{Q}}$
6: $\chi = l_{\mathbb{Q}} \cdot \ln 2 + k_{\mathbb{Q}}$
7: $\psi = 1 + \sum_{i=1}^N \chi^i \cdot f_i$
8: return $\big((\psi \cdot \alpha_e) << l_{\mathbb{Z}}\big)_{\mathbb{Z}}$
```

The exponential function takes, as its fundamental input, a uniformly sampled $x \in \beta_0^+$ and returns a $y \in [0, 1)$, to λ bits of precision. This y will be represented as a fraction over 2^λ, or more precisely, as a λ-bit extended unsigned integer type with the implied denominator having been accounted for by the operations which act on x. There are three steps: (i) Calculate shift and input to Taylor series, (ii) Evaluate the Taylor series and iii) apply shift to the result of the Taylor series.

Let f_i be an approximation, to p bits of precision, of $1/i!$. Hence, the fixed-point Taylor series is given by

$$y = \sum_{i=1}^{n} \chi^i \cdot f_i. \tag{12}$$

Because of the propagation of uncertainty through operations on finite representations of numbers in $\mathbb{R}$, the constants (such as $\ln 2$, the inverse factorials, etc.) are required to have greater precision than the output precision, λ.

Theorem 2. *The precision, p, to which χ and the set of f_i must be stored is given by*

$$p = \lambda + \log_2 \Big(\sum_{i=1}^{n} |i \cdot \chi^{i-1} \cdot f_i| + |\chi^i| \Big). \tag{13}$$

Proof. As $y_\chi = \sum_{i=1}^n \chi^i \cdot f_i$, and has λ bits of precision, the input value χ and the factorial constants, f_i, will be required to have p bits of precision such that $\delta\chi = 2^{-(p+1)}$ and $\delta f_i = 2^{-(p+1)}$. From this it is required that

$$\delta\Big(\sum_{i=1}^{n} \chi^i \cdot f_i \Big) \leq 2^{-(\lambda+1)}. \tag{14}$$

Uncertainty propagates through this expression in the following ways

$$\frac{\delta\chi^i}{|\chi^i|} = |i| \cdot \frac{\delta\chi}{|\chi|}, \quad \frac{\delta(\chi^i \cdot f_i)}{|\chi^i \cdot f_i|} = \frac{\delta\chi^i}{|\chi|} + \frac{\delta f_i}{|f_i|} \tag{15}$$

and, hence,

$$\delta\Big(\sum_{i=1}^{n}\chi^i\cdot f_i\Big) = \sum_{i=1}^{n}\delta(\chi^i\cdot f_i) \tag{16}$$

$$= \sum_{i=1}^{n}\left(\frac{\delta\chi^i}{|\chi|}+\frac{\delta f_i}{|f_i|}\right)\cdot|\chi^i\cdot f_i| \tag{17}$$

$$= \sum_{i=1}^{n}\left(|i|\cdot\frac{\delta\chi}{|\chi|}+\frac{\delta f_i}{|f_i|}\right)\cdot|\chi^i\cdot f_i| \tag{18}$$

Substituting in the required uncertainties in terms of λ and p and rearranging gives

$$2^{-(\lambda+1)} = 2^{-(p+1)}\cdot\sum_{i=1}^{n}\Big(|i\cdot\chi^{i-1}\cdot f_i|+|\chi^i|\Big) \tag{19}$$

and

$$p = \lambda+\log_2\Big(\sum_{i=1}^{n}|i\cdot\chi^{i-1}\cdot f_i|+|\chi^i|\Big). \tag{20}$$

■

Note that the $i = 0$ term, which goes to 1, contributes nothing to the error and has furtively disappeared from the analysis. Equation (20) gives the precision to which the inverse factorials must be stored and a similar analysis on the constants used before the Taylor Series shows that, in total, 32 extra bits would suffice. The reference implementation uses an extra 64 bits for maintaining simplicity in the arbitrary precision arithmetic, so the algorithm can be further optimised for performance as the Taylor series is the bottleneck of the Gaussian function and sensitive to the size of the input.

The number of terms, n, is small for values close to the point around which a Taylor expansion was taken, $x = 0$ in this case. As this algorithm must exit all iterations as if it were the worst case, n and, hence, χ must not grow large. Equation (20) is monotonically increasing, but grows to only 2 extra bits for χ between 0 and 1, whereas for χ approaching 2, the required extra bits is above 40. This amounts to extra storage required for the inverse factorials and overhead in the most computationally expensive part of the algorithm, in dealing with the non-zero, increasing integer components χ^i.

The potential overflow from converting between base 2 and base e to get Equation (9) is to be avoided and we choose to allow χ to overflow or underflow, keeping track of this with a selective multiplication by either 1, $1/e$ or e. We propose a constant-time solution to this issue with the final Gaussian function defined in Algorithm 5.

The constant-time underflow and overflow operations adhere to the same logical conventions as described in Section 4.1. The function `ct_lt` is a constant-time $<$ operation and `ct_select` is the same as before, although it is now used twice in succession to select between 1, e or $1/e$.

Algorithm 5 $\textsc{Gauss\_Exp}(x, s, \lambda)$

```
1: Require: $\{f_i = \frac{1}{i!}\}_{i=1}^{N}$    ▷ $N$ s.t. $f_{N+1} < 2^{-\lambda}$
2: $x'_{\mathbb{Z}} = (s \cdot x^2)_{\mathbb{Z}}$
3: $x'_{\mathbb{Q}} = (s \cdot x^2)_{\mathbb{Q}}$
4: $l_{\mathbb{Z}} = \left(\log_2 e \cdot (\lambda \cdot \ln 2 - x'_{\mathbb{Z}})\right)_{\mathbb{Z}}$
5: $l_{\mathbb{Q}} = \left(\log_2 e \cdot (\lambda \cdot \ln 2 - x'_{\mathbb{Z}})\right)_{\mathbb{Q}}$
6: $\chi = l_{\mathbb{Q}} \cdot \ln 2 - x'_{\mathbb{Q}}$
7: $b_e = 0$    ▷ Let $b_e$ be unsigned.
8: $b_e$ −= ct_underflow($\chi$, $l_{\mathbb{Q}} \cdot \ln 2$, $x'_{\mathbb{Q}}$)
9: $t = \chi$
10: $\chi$ += $(\lambda \cdot \ln 2)_{\mathbb{Q}}$
11: $b_e$ += ct_overflow($\chi$, $t$, $(\lambda \cdot \ln 2)_{\mathbb{Q}}$)
12: $\psi = 1 + \sum_{i=1}^{N} \chi^i f_i$
13: $c_e$ = ct_lt($0$, $b_e$)$\wedge$ ct_lt($b_e$, $0 - 1$)    ▷ 1 if $b_e = 1$
14: $\alpha_e$ = ct_select($1$, $e$, $c_e$)
15: $c_e$ = ct_lt($1$, $c_e$)
16: $\alpha_e$ = ct_select($\alpha_e$, $1/e$, $c_e$)
17: return $\left((\psi \cdot \alpha_e) << l_{\mathbb{Z}}\right)_{\mathbb{Z}}$
```

Listing 1[1] shows the code for the constant-time operations used in the reference implementation of the Ziggurat sampler. The `UINT` types are the standard unsigned integers prefixed by whichever number of bits they have. The `fix _t` types are also labelled by their bit precision and represent fixed point fractions composed of a number of `UINT64` types. For example, if `n` is a `fix128_t`, it will contain two `UINT64` types in a struct, called `n.a0` and `n.a1`. The logical functions return a 0 or 1 and the selection functions return the selected value.

```
UINT32 ct_isnonzero_f128(fix128_t x)
{
  return ((x.a0|-x.a0) >> 63) & ((x.a1|-x.a1) >> 63);
}

UINT32 ct_isnonzero_u32(UINT32 x)
{
  return (x|-x)>>31;
}

UINT32 ct_lt_u32(UINT32 x, UINT32 y)
{
  return (x^((x^y)|((x-y)^y)))>>31;
}

UINT32 ct_lt_u64(UINT64 x, UINT64 y)
```

[1] These functions are adapted from `https://cryptocoding.net/index.php/Coding_rules` and have been extended to use multi-precision logic.

```
{
  return (x^((x^y)|((x-y)^y)))>>63;
}

UINT32 ct_lte_u32(UINT32 x, UINT32 y)
{
  return 1 ^ ((y^((y^x)|((y-x)^x)))>>31);
}

UINT32 ct_lte_f128(fix128_t a, fix128_t b)
{
  return ct_lt_u64(a.a1, b.a1) |
         ct_select_64(0, (1^ct_lt_u64(b.a0, a.a0)),
             (1^((a.a1-b.a1)|(b.a1-a.a1))>>63));
}

UINT32 ct_neq_u32(UINT32 x, UINT32 y)
{
    return ((x-y)|(y-x))>>63;
}

UINT32 ct_select_u32 (UINT32 a, UINT32 b, UINT32 bit)
{
  /* -0 = 0, -1 = 0xff....ff */
  UINT32 mask = - bit;
  UINT32 ret = mask & (a^b);
  ret = ret ^ a;
  return ret;
}

fix256_t ct_select_f256 (fix256_t a, fix256_t b, UINT64 bit)
{
  /* -0 = 0, -1 = 0xff....ff */
  UINT64 mask = - bit;
  fix256_t ret;

  ret.a0 = mask & (a.a0 ^ b.a0);
  ret.a1 = mask & (a.a1 ^ b.a1);
  ret.a2 = mask & (a.a2 ^ b.a2);
  ret.a3 = mask & (a.a3 ^ b.a3);

  ret.a0 = ret.a0 ^ a.a0;
  ret.a1 = ret.a1 ^ a.a1;
  ret.a2 = ret.a2 ^ a.a2;
  ret.a3 = ret.a3 ^ a.a3;
  return ret;
}
```

Listing 1: Constant-time operations to the various precisions required for a 128-bit implementation of the Ziggurat sampler.

5 Results

This section discusses the enhancements to the Ziggurat method provided by our algorithm. In particular, the low-level construction of the sampler leads to a significant reduction in the memory footprint, as presented in Section 5.1, and, as outlined in Section 5.2, the side-channel resilience of our algorithm makes the Ziggurat method, and the range of parameters to which it is suited (i.e. high standard deviation), a more attainable objective for LBC.

5.1 Performance and validation

The algorithm presented in this paper solves issues involved with sampling from the discrete Gaussian distribution over the integers via the Ziggurat method, with significantly better resilience to side-channel attacks. The sampler retains its efficiency, improves upon use of memory resources and is more suitable for application to low-memory devices and hardware due to the integer arithmetic and lack of division.

Section 5.1 shows the performance and memory profiles of our proposed sampler, as well as the original Ziggurat and the CDT [17] samplers. We refer to our sampler as Ziggurat_O and to the original algorithm, proposed by Buchmann et al. [13], as Ziggurat_B. We notice only a slight decrease in performance, accompanied by improvements of orders of magnitude in memory use, especially when code is taken into account (as can be seen by the sizes of executables). It should be noted, however, that the reference implementation was built with functionality in mind, and there is room for optimising the code, see Section 4.1. The

Sampler	Time (ms for 10^6 samples)	Stack and Heap Allocations (Max) (B)	Size of executable (B)
Ziggurat_O	1,102	1,200	27,376
Ziggurat_B	1,012	123,000	2,036,608
CDT	320	5,961,000	45,576

Table 2: Performance and memory profile of 10^6 samples at $\sigma = 19600$ for our sampler, Ziggurat_O, Buchmann et al.'s sampler, Ziggurat_B, and the CDT sampler [17]. All measurements were made with a single CPU on an an Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz. Note, the number of rectangles was 64.

results show significant improvements in the memory consumption of the Ziggurat sampler. It should be noted that the CDT algorithm has been optimised for both efficiency and memory, as it is a core component of the *Safecrypto* library [17]. For example, the full table sizes of the cumulative distribution function for $\sigma = 19600$ is a few times the value given here. The table sizes have been decreased using properties of the Kullback-Leibler divergence of the Gaussian distribution [18]. The Ziggurat's memory profile is orders of magnitude better than that of the CDT and its performance is a small factor slower. With algorithmic ideas for increasing performance suggested in Section 4.1, alongside low-level optimisations already applied to the CDT sampler (e.g. struct packing), we expect the small factor by which the performance drops can be reduced, possibly to the extent of becoming a performance gain.

For qualitative assurance of functionality, see Figure 3 which shows the frequency distributions for 10^8 samples for Buchmann's sampler and that proposed in this paper. The sampler behaves as expected, producing a discrete Gaussian distribution at high standard deviation.

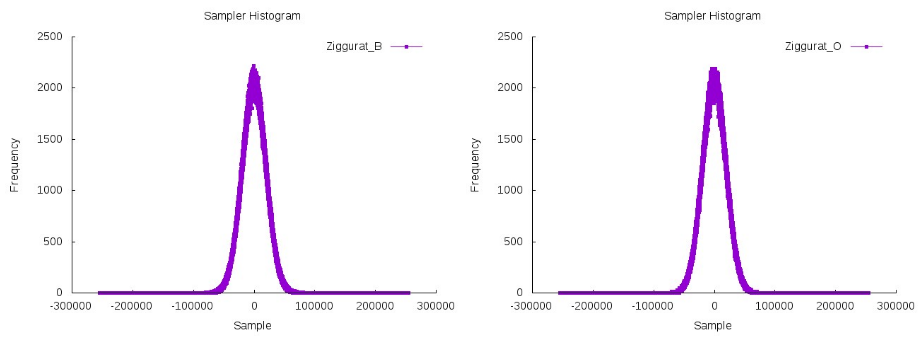

Fig. 3: Histograms obtained from 10^8 samples of the two Ziggurat algorithms.

5.2 Side Channel Security

We referred to a possible attack on the unmodified Ziggurat sampler in Section 4.1, where the $x = 0$ sample is readily obtained by the difference in timing of the logic in Line 6 in Algorithm 1 to every other sample. It is seemingly not mentioned elsewhere in the literature. Furthermore, most implementations of the exponential function are not constant-time and will perform the approximation over a given, low valued, domain and raise it to a power dependent on how large the initial exponent was. Large lookup tables are often used to achieve high performance and, should the exponent match a member exactly, the worst-case scenario is direct leakage of samples through any timing method.

Typical side-channel protections to timing attacks involve ensuring that operations which depend on secret data are done in constant time. This is, seemingly, impossible for a rejection sampler. For the Ziggurat sampler, limiting to two possible paths from beginning to accept/reject is, hence, the best that can be done. It is important, however, that all elements of the sample space can be found to have been sampled via both accept paths, which is the case for the enhanced Ziggurat.

Further to the more general timing attacks, the "Flush, Gauss and Reload" attack [3] is a topic of on-going research for which the solutions must be tested on the Ziggurat method. This paper presents an attack on the Gaussian samplers of the BLISS signature scheme [4], but also provides unique countermeasures for each sampling method. Fitting these countermeasures individually and assessing the impact on performance is beyond the scope of this paper, but the authors of the cache attack have discussed how the Ziggurat's countermeasures have significantly less overhead, in theory, than than those of the CDT and Knuth/Yao.

The attack can be summarised as follows. Any non-uniformity in the accessing of table elements can lead to cache misses and timing leakage. It requires that the attacker have shared cache space, which is not typical of constrained systems, but also not an impossible situation. The countermeasure for the Ziggurat sampler amounts to ensuring a `load` operation is called on all rectangle points, regardless of whether they are needed. The data is loaded but not used further in most cases.

General solutions also exist to counter this attack. One such solution was proposed by Roy [19], whereby the samples are shuffled and the particular samples for which a timing difference can be made are obscured. An analysis of the shuffling method was carried out by Pessl [20] and improvements were made, but research into the effect of these on the performance and memory profile of samplers is, also, on-going.

Despite the uncertainty surrounding this attack, and the performance penalties induced by the suggested solutions, we expect that the sampler proposed in this paper will not be impacted negatively under the imposed constraints of the "Flush, Gauss and Reload" attack. It is suggested by the authors of the paper that the Knuth/Yao and CDT samplers be implemented in constant time to counter the attack. In contrast, the countermeasures for the Ziggurat sampler amount to two more (blind) load operations with every sample, which is both negligible compared to the operations already being performed and significantly less expensive than implementing the Ziggurat in constant time. We argue, however, that the sampler is required to be secure against attacks from direct timing measurements of operations, before countermeasures against cache attacks can be facilitated.

6 Conclusion

We proposed a discrete Gaussian sampler using the Ziggurat method, which significantly negates its vulnerability to side-channel cryptanalysis. Our research improves the Ziggurat sampler's memory consumption by more than a factor of 100 and maintains its efficiency under the new security constraints. Compared with the CDT sampler, the Ziggurat is nearly 5,000 times less memory-intensive. A significant amount of work has been carried out on making the sampler more portable and lightweight, as well as less reliant on hardware or software features, such as floating-point arithmetic and extended precision integers. The result is a sampler which is notably more suitable for use in industry, for its portability and lack of dependencies, and as a research tool, for its self-contained implementation of the low-level components which make up the entire sampler.

References

1. C. Peikert, "A decade of lattice cryptography," *Foundations and Trends in Theoretical Computer Science*, vol. 10, no. 4, pp. 283–424, 2016. [Online]. Available: http://dx.doi.org/10.1561/0400000074
2. D. Micciancio and O. Regev, "Worst-case to average-case reductions based on Gaussian measures," in *45th Annual IEEE Symposium on Foundations of Computer Science*, Oct. 2004, pp. 372–381.
3. L. G. Bruinderink, A. Hülsing, T. Lange, and Y. Yarom, "Flush, gauss, and reload–a cache attack on the BLISS lattice-based signature scheme," in *International Conference on Cryptographic Hardware and Embedded Systems.* Springer, 2016, pp. 323–345.
4. L. Ducas, A. Durmus, T. Lepoint, and V. Lyubashevsky, "Lattice signatures and bimodal Gaussians," in *Advances in Cryptology–CRYPTO 2013.* Springer, 2013, pp. 40–56.
5. N. Genise and D. Micciancio, "Faster Gaussian sampling for trapdoor lattices with arbitrary modulus," Cryptology ePrint Archive, Report 2017/308, 2017, https://eprint.iacr.org/2017/308.
6. L. Chen, S. Jordan, Y.-K. Liu, D. Moody, R. Peralta, R. Perlner, and D. Smith-Tone, *Report on post-quantum cryptography.* US Department of Commerce, National Institute of Standards and Technology, 2016.
7. J. Hoffstein, J. Pipher, W. Whyte, and Z. Zhang, "pqntrusign: update and recent results," 2017. [Online]. Available: https://2017.pqcrypto.org/conference/slides/recent-results/zhang.pdf
8. Z. Zhang, C. Chen, J. Hoffstein, and W. Whyte, "NTRUEncrypt," Technical report, National Institute of Standards and Technology, 2017. available at https://csrc. nist. gov/projects/post-quantum-cryptography/round-1-submissions, Tech. Rep., 2017.
9. T. H. Le Trieu Phong, Y. Aono, and S. Moriai, "Lotus," Technical report, National Institute of Standards and Technology, 2017. available at https://csrc. nist. gov/projects/post-quantum-cryptography/round-1-submissions, Tech. Rep., 2017.
10. C. Peikert, "An efficient and parallel Gaussian sampler for lattices," in *International Cryptoology Conference CRYPTO 2010*, ser. CRYPTO '10. Santa Barbara, CA, USA: Springer Berlin Heidelberg, 2010.
11. S. Sinha Roy, F. Vercauteren, and I. Verbauwhede, "High precision discrete Gaussian sampling on FPGAs," in *Selected Areas in Cryptography – SAC 2013*, T. Lange, K. Lauter, and P. Lisoněk, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 383–401.
12. D. Micciancio and M. Walter, "Gaussian Sampling over the Integers: Efficient, Generic, Constant-Time," Tech. Rep. 259, 2017. [Online]. Available: https://eprint.iacr.org/2017/259
13. J. Buchmann, D. Cabarcas, F. Göpfert, A. Hülsing, and P. Weiden, "Discrete Ziggurat: A Time-Memory Trade-Off for Sampling from a Gaussian Distribution over the Integers," in *Selected Areas in Cryptography – SAC 2013*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, pp. 402–417, dOI: 10.1007/978-3-662-43414-7_20. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-662-43414-7_20

14. V. Shoup, “Number theory c++ library (ntl) version 10.3.0,” http://www.shoup.net/ntl, 2003.
15. GNU, “glibc-2.7,” https://www.gnu.org/software/libc/, 2018.
16. G. Marsaglia and W. W. Tsang, “The Ziggurat Method for Generating Random Variables,” *Journal of Statistical Software*, vol. 5, no. 1, pp. 1–7, 2000. [Online]. Available: https://www.jstatsoft.org/index.php/jss/article/view/v005i08
17. “libsafecrypto: WP6 of the SAFEcrypto project - a suite of lattice-based cryptographic schemes,” Jul. 2018, original-date: 2017-10-16T14:56:31Z. [Online]. Available: https://github.com/safecrypto/libsafecrypto
18. T. Pöppelmann, L. Ducas, and T. Güneysu, “Enhanced Lattice-Based Signatures on Reconfigurable Hardware,” in *Cryptographic Hardware and Embedded Systems CHES 2014*, ser. CHES ’14. Busan, South Korea: Springer Berlin Heidelberg, 2014, pp. 353–370.
19. S. S. Roy, O. Reparaz, F. Vercauteren, and I. Verbauwhede, “Compact and side channel secure discrete gaussian sampling.” *IACR Cryptology ePrint Archive*, vol. 2014, p. 591, 2014.
20. P. Pessl, “Analyzing the shuffling side-channel countermeasure for lattice-based signatures,” in *International Conference in Cryptology in India*. Springer, 2016, pp. 153–170.