On Practical Discrete Gaussian Samplers for Lattice-Based Cryptography

Howe, J., Khalid, A., Rafferty, C., Regazonni, F., & O'Neill, M. (2016). On Practical Discrete Gaussian Samplers for Lattice-Based Cryptography. *IEEE Transactions on Computers*. Advance online publication. https://doi.org/10.1109/TC.2016.2642962

Published in:
IEEE Transactions on Computers

Document Version:
Peer reviewed version

Queen's University Belfast - Research Portal:
Link to publication record in Queen's University Belfast Research Portal

Publisher rights
© 2016 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

General rights
Copyright for the publications made accessible via the Queen's University Belfast Research Portal is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The Research Portal is Queen's institutional repository that provides access to Queen's research output. Every effort has been made to ensure that content in the Research Portal does not infringe any person's rights, or applicable UK laws. If you discover content in the Research Portal that you believe breaches copyright or violates any law, please contact openaccess@qub.ac.uk.

Open Access
This research has been made openly available by Queen's academics and its Open Research team. We would love to hear how access to this research benefits you. – Share your feedback with us: http://go.qub.ac.uk/oa-feedback

Download date:29. May. 2024

On Practical Discrete Gaussian Samplers For Lattice-Based Cryptography

James Howe, Ayesha Khalid, Ciara Rafferty, *Member, IEEE*, Francesco Regazzoni, *Member, IEEE*, & Máire O'Neill, *Senior Member, IEEE*

Abstract—Lattice-based cryptography is one of the most promising branches of quantum resilient cryptography, offering versatility and efficiency. Discrete Gaussian samplers are a core building block in most, if not all, lattice-based cryptosystems, and optimised samplers are desirable both for high-speed and low-area applications. Due to the inherent structure of existing discrete Gaussian sampling methods, lattice-based cryptosystems are vulnerable to side-channel attacks, such as timing analysis. In this paper, the first comprehensive evaluation of discrete Gaussian samplers in hardware is presented, targeting FPGA devices. Novel optimised discrete Gaussian sampler hardware architectures are proposed for the main sampling techniques. An independent-time design of each of the samplers is presented, offering security against side-channel timing attacks, including the first proposed constant-time Bernoulli, Knuth-Yao, and discrete Ziggurat sampler hardware designs. For a balanced performance, the Cumulative Distribution Table (CDT) sampler is recommended, with the proposed hardware CDT design achieving a throughput of 59.4 million samples per second for encryption, utilising just 43 slices on a Virtex 6 FPGA and 16.3 million samples per second for signatures with 179 slices on a Spartan 6 device.

1 Introduction

Essentially all asymmetric cryptographic primitives used for secure Internet communications, such as RSA and ECDSA, will be rendered completely insecure with the practical development of a quantum computer, by virtue of Shor's algorithm [37]. Even symmetric-key encryption schemes, such as the advanced encryption standard (AES), will have a quadratic brute-force speed-up via Grover's algorithm [13], [14] – decreasing a search space of $\mathcal{O}(2^n)$ to $\mathcal{O}(2^{n/2})$ –making AES-128 as secure as AES-64. There are now emerging branches of cryptography resistant to these quantum reductions, namely, quantum-resilient or post-quantum cryptography.

Post-quantum cryptography deals with non-quantum operations but is theoretically strong against cryptanalysis on classical and quantum computers. Recently, post-quantum cryptography has seen significant advancements in many areas, due to progress in advanced frameworks such as the Internet of Things as well as the need for long-term, highly secure, quantum-resilient cryptographic primitives. Such progress has led to growth in novel cryptographic constructions such as encryption schemes and signature schemes, as well as more advanced schemes such as fully homomorphic encryption, which can process on encrypted data and can be used to secure practical cloud infrastructures. Furthermore, with the announcement that the governmental agencies are planning transitions towards quantum-resistant algorithms [6], with CESG focusing on post-quantum cryptography as opposed to quantum technologies [5], it is clear how important these drop-in replacements are for classical cryptosystems.

Lattice-based cryptography (LBC) [1], [32], a subtype of post-quantum cryptography, bases its hardness assumption on finding the shortest (or closest) vector in a lattice, which is to date resilient to attacks by a quantum computer. LBC is very promising as it offers extended functionality whilst being more efficient than ECC and RSA based primitives of public-key encryption [31] and digital signature schemes [28], [16], at the cost of larger key sizes. Lattice-based cryptoschemes are usually founded on either the learning with errors problem (LWE) [32] or the short integer solution problem (SIS) [1], which are based on standard lattices, or their equivalent ring variants ring-LWE and ring-SIS [27], [20], [23]. These ring-variants enable the use of number theoretic transform (NTT) operations for polynomial multiplication, as opposed to matrix-vector multiplications, resulting in significant savings in computational efficiency. For nearly a decade the concepts remained solely academic, due to inefficiencies such as large key sizes. However, LBC primitives are now viable for practical implementation (see the FPGA design [28] of BLISS [10]) and many compete well with equivalent, classical schemes.

As LBC starts to become feasible for being deployed in the real world, suitable countermeasures against *physical cryptanalysis*, including side-channel analysis (SCA), become a requirement, which is echoed by a recent call by NIST for quantum-resistant algorithms resistant to SCA attacks [25]. The key building blocks in LBC include matrix-vector multiplication for standard lattice schemes, polynomial multiplication for ideal lattice schemes, and

J. Howe, A. Khalid, C. Rafferty, and M. O'Neill are with the Centre for Secure Information Technologies (CSIT), Queen's University Belfast, Northern Ireland (e-mail: {jhowe02, a.khalid, c.m.rafferty, maire.oneill}@qub.ac.uk).
F. Regazzoni is with the Advanced Learning and Research Institute, Università della Svizzera Italiana, Switzerland (email: regazzoni@alari.ch).

discrete Gaussian sampling. Arguably the most vulnerable module within modern lattice-based constructions is the discrete Gaussian sampler, which, when successfully attacked, renders the whole cryptosystem broken [3]. To date, there has been little research into the SCA-resilience of lattice-based cryptographic implementations (with only Roy *et al.* [34], [33] as a preface). The rationale for employing discrete Gaussian samplers (as opposed to another probability distribution) is that they allow for more efficient implementations, with smaller output sizes such as ciphertexts or signatures. Moreover, they are used to add "noise" onto values that would otherwise give away secret information via Gaussian elimination. They are, however, highly susceptible to timing analysis attacks due to their inherent non-constant run-time (due to the normalised structure and/or speed optimisations).

This research proposes practical and time-independent hardware implementations of all discrete Gaussian samplers used in lattice-based encryption and signature schemes[1], that is, the Bernoulli sampler [10], the cumulative distribution table (CDT) sampler [26], the Knuth-Yao sampler [17], and the discrete Ziggurat sampler [22], [4]. The major contributions of this research are as follows:

1) The first comprehensive evaluation of all aforementioned discrete Gaussian samplers, which are a core component of lattice-based encryption and signature schemes, with mathematical descriptions of all prominent sampling techniques, discussions of their inherent limitations and strengths, as well as previous comparable hardware implementations.
2) Practical hardware FPGA designs of discrete Gaussian samplers are presented, compared with current state-of-the-art implementations, for appropriate practical parameters, throughput, memory consumption, and resource count. Novel optimisation strategies are proposed for the hardware architectures of the sampling techniques, where the results compete with, and in many cases, significantly outperform, previous implementations.
3) The proposed hardware designs operate in independent-time and therefore provide resistance against timing analysis attacks.
4) Based on the performance results, recommendations are given for the most appropriate sampler to use in particular applications.

The paper is outlined as follows: firstly, prerequisites are given on the discrete Gaussian distribution. Section 3 presents previous work on discrete Gaussian sampling techniques. Section 4 addresses the desirable features each sampler requires to operate in constant-time, and in Section 5 the hardware architectures of each constant-time discrete Gaussian sampler are described. Section 6 presents the performance results of the samplers, compared to previous implementations, and targeted recommendations for specific implementations are given.

[1]With the Binomial sampler used in [2] currently only applicable for key-exchange protocols.

2 Prerequisites/Preliminaries

2.1 The Discrete Gaussian Distribution

The *discrete Gaussian distribution* or *discrete normal distribution* $D_{\mathbb{Z},\sigma}$ over $\mathbb{Z}$ with mean 0 and standard deviation σ is defined to have a weight proportional to $\rho_\sigma(x) = \exp(\frac{-x^2}{2\sigma^2})$ for all integers $x \in \mathbb{Z}$. Considering $S_\sigma = \rho_\sigma(\mathbb{Z}) = \sum_{k=-\infty}^{\infty} \rho_\sigma(k) \approx \sqrt{2\pi}\sigma$ the probability of sampling $x \in \mathbb{Z}$ from the distribution $D_{\mathbb{Z},\sigma}$ is calculated as $\rho_\sigma(x)/S_\sigma$. In this research, the standard deviation σ is assumed fixed, thus, it is sufficient 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 used to output values over $\mathbb{Z}$.

2.2 Practical Discrete Gaussian Parameters

The parameters for discrete Gaussian sampling depend significantly on the application. To appropriately evaluate the performance of the samplers, two of the most common parameter sets are considered. These are encryption parameters from Lindner and Peikert [19] (LP)[2] with $\sigma = 3.33$ (denoted as σ_{LP}) and digital signature parameters from Ducas *et al.* [10] (BLISS) with $\sigma = 215$ (denoted as σ_{BLISS}); this choice allows for a more comprehensive analysis, covering both encryption and digital signature schemes. Table 1 shows the parameters required for discrete Gaussian sampling in practice.

Due to the nature of the discrete Gaussian distribution, that is with infinitely long tails and infinitely high precision, it is essential to make practical compromises which also do not hinder the integrity of the scheme. The discrete Gaussian parameters needed are (σ, λ, τ), representing the sampler's standard deviation, precision and tail-cut respectively, and are detailed as follows:

- ◇ The standard deviation (σ) controls the distribution's shape by quantifying the dispersion of data from the mean. The standard deviation depends on the modulus used within in the LWE or SIS scheme. For instance in LWE, should σ be too small the hardness assumption may become easier than expected, and if σ is too large the problem may not be as well-defined as required.
- ◇ The precision parameter (λ) governs the level of precision required for an implementation, exacting the statistical distance between the "perfect" theoretical discrete Gaussian distribution and the "practical" to be no greater than $2^{-\lambda}$, corresponding directly to the scheme security level. Saarinen [36] recommends that for a scheme with target security level λ-bits, precision need be no greater than $\lambda/2$, arguing that there exists *no algorithm* that can distinguish between a "perfect" sampler and one with statistical distance $2^{-\lambda/2}$.
- ◇ The tail-cut parameter (τ) administers how much of the less-heavy tails can be excluded in the practical implementation, for a given security level. That is, given a target security level of s-bits, the target distance from "perfect" need be no less than 2^{-s}. Therefore, instead of

[2]The same parameters are used for implementing the ring-LWE encryption scheme of Lyubashevsky *et al.* [21], see [31] for a hardware implementation.

considering samples in the range $|x| \in \{0, \infty\}$, they can be considered in the range $|x| \in \{0, \sigma\tau\}$. Applying the reduction in precision also affects the tail-cut parameter, which is calculated as $\tau = \sqrt{\lambda \times 2\ln(2)}$.

Parameters	(σ, λ, τ)
LP Encryption [19]	$(3.33, 64, 9.42)$
BLISS Signatures [10]	$(215, 64, 9.42)$

TABLE 1: Secure 128-bit discrete Gaussian parameters.

2.3 Gaussian Convolution

Further reductions in memory are possible by virtue of Peikert's convolution lemma [24], adapted by Pöppelmann *et al.* [28] using the Kullback-Leibler divergence. Referring to [26], [28] for the formal definitions of the *smoothing parameter* η and Kullback-Leibler divergence respectively, the adaption in [28] states:

Lemma 1: Let $x_1 \leftarrow D_{\mathbb{Z},\sigma_1}$, $x_2 \leftarrow D_{k\mathbb{Z},\sigma_2}$ for some positive real σ_1, σ_2 and let $\sigma_3^{-2} = \sigma_1^{-2} + \sigma_2^{-2}$ and $\sigma^2 = \sigma_1^2 + \sigma_2^2$. For any $\epsilon \in (0, \frac{1}{2})$ if $\sigma_1 \geq \eta_\epsilon(\mathbb{Z})/\sqrt{2\pi}$ and $\sigma_3 \geq \eta_\epsilon(k\mathbb{Z})/\sqrt{2\pi}$, then ("perfect") distribution $\mathcal{P}$ of $x_1 + x_2$ verifies

$$D_{\mathrm{KL}}(\mathcal{P}||D_{\mathbb{Z},\sigma}) \leq 2\Big(1 - \Big(\frac{1+\epsilon}{1-\epsilon}\Big)^2\Big)^2 \approx 32\epsilon^2.$$

Proof: The proof of this lemma is referred to in [28]. □

Utilising Lemma 1 minimises the standard deviation $\sigma_{\mathrm{BLISS}} = 215$. For BLISS, Lemma 1 is satisfied by setting $k = 11$, which means $\sigma' = \sigma/\sqrt{1+k^2} \approx 19.47$, and by sampling twice $x_1', x_2' \leftarrow D_{\mathbb{Z},\sigma'}$ a value $x \leftarrow D_{\mathbb{Z},\sigma}$ can be built as $x = x_1' + kx_1'$. The usage of the significantly smaller σ'_{BLISS} means that sizes of precomputed tables within the sampling techniques are reduced to around 11x smaller, with the requirement of sampling twice.

3 Sampling Techniques – Background & Previous Work

Lattice-based cryptoschemes, based on LWE and SIS problems, require discrete Gaussian noise to mask the scheme's secret key. In this section, the main techniques for generating discrete Gaussian noise are surveyed.

3.1 Rejection Sampling

Using *rejection sampling* or the *acceptance-rejection method* [39], it is possible to sample from an arbitrary target distribution f when given access to a bounded probability distribution g. To sample from f, a sample from g is accepted with probability $f(x)/(M \cdot g(x))$, where M is some positive real. When $f(x) \leq M$ for all x, the sampler produces the exact distribution f. To use rejection sampling to draw values from the positive half of a discrete Gaussian distribution, a uniformly random integer $x \in \{0, ..., \tau\sigma\}$ is chosen and accepted with a probability proportional to $\rho_\sigma(x) = \exp(-x^2/2\sigma^2)$ (in this case $M = 1$). A uniformly random value x is selected, a random value u chosen from $[0, 1)$ and it is checked whether $u < \rho_\sigma(x)$ and thus whether a random u is under (acceptance) or above the curve (rejection) of the probability mass function of the Gaussian distribution. Sampling from the full range of the discrete Gaussian distribution involves rejection of an accepted $x = 0$ with probability $\frac{1}{2}$ and sampling of a sign bit. On average the method requires $2\tau/\sqrt{2\pi}$ trials until a sample is accepted. As this method requires many rejections to achieve an accepted value (an average of ≈ 8 trials) and the costly computation of the exponential function to high precision, rejection sampling is considered inefficient for practical hardware instantiations.

3.2 Bernoulli Sampling

The approach in Section 3.1 can be optimised to reduce the amount of rejections. This is achieved in the scheme by Ducas *et al.* [10] (Figure 1), where Bernoulli distributed variables $\mathcal{B}_c$ are used, outputting one with probability $c \in [0, 1]$, and zero otherwise. The number of rejections are reduced by using a distribution g called binary Gaussian distribution where x is proportional to $\rho_{\sigma_{\mathrm{bin}}}(x) = 2^{-x^2}$ with parameter $\sigma_{\mathrm{bin}} = \sqrt{1/(2\ln 2)} \approx 0.849$ (see Algorithm 1) which can be constructed on-the-fly. Using the binary Gaussian, an intermediate distribution $k \cdot D_{\mathbb{Z}^+,\sigma_{\mathrm{bin}}} + \mathcal{U}(\{0 \ldots k-1\})$ is constructed and the correct Gaussian distribution $D_{\mathbb{Z}^+,k\sigma_{\mathrm{bin}}}$ is shaped using rejection sampling (Algorithm 2), which reduces the number of rejections to ≈ 1.47 (compared to 8 for classical rejection sampling). For rejection sampling, a value is accepted with probability $f(x)/(M \cdot g(x))$ which usually requires explicit computation of $f(x)$. However, it holds for an integer $x = \sum_{i=0}^{\ell-1} x_i 2^i$ with $x_i \in \{0, 1\}$ that:

$$\mathcal{B}_{\exp(-x/f)} = \mathcal{B}_{\exp\left(-\sum_i x_i 2^i/f\right)} =$$
$$\mathcal{B}_{\prod_i \exp(-x_i 2^i/f)} = \bigwedge_{i \text{ s.t. } x_i = 1} \mathcal{B}_{\exp(-2^i/f)},$$

meaning the final rejection step is performed independently, by evaluation of Bernoulli trials, which is efficient given precomputed biases c_i for every x_i. The method only requires $\lambda \log_2(2.4\tau\sigma^2)$ bits of storage, and it is shown in [10] that Algorithm 2 requires less than $1.47 + 1/k$ trials. Algorithm 3 then corrects z to fit the final target distribution $D_{\mathbb{Z},k\sigma_{\mathrm{bin}}}$. One caveat to the Bernoulli technique is that the standard deviation must be a multiple of the binary Gaussian standard deviation; that is $\sigma = k\sigma_{\mathrm{bin}}$ where $k = 4$ for LP, $k = 254$ for BLISS, and $\sigma_{\mathrm{bin}} = \sqrt{1/2\ln 2}$. Thus, the standard deviations (Table 1) become $\sigma_{\mathrm{LP}} = 3.39$, $\sigma'_{\mathrm{BLISS}} = 19.53$, and $\sigma_{\mathrm{BLISS}} = 215.73$.

Bernoulli sampling has been implemented without using the binary Gaussian distribution [31] for the small standard deviation necessary for lattice-based public-key encryption. A complete hardware implementation of the sampler can be found in the BLISS signature scheme by Pöppelmann *et al.* [28], where the binary Gaussian distribution $\rho_{\sigma_{\mathrm{bin}}}(\{0, 1, \ldots, j\}) = \sum_{i=0}^{j} 2^{-i^2} =$

Algorithm 1 Sampling $D_{\mathbb{Z}^+,\sigma_{\text{bin}}}$

Ensure: An integer $x \in \mathbb{Z}^+$ according to $D^+_{\sigma_{\text{bin}}}$
  Generate a bit $b \leftarrow \mathcal{B}_{1/2}$
  if $b = 0$ **then** return 0
  for $i = 1$ to ∞ **do**
    draw random bits $b_1 \dots b_k$ for $k = 2i - 1$
    if $b_1 \dots b_{k-1} \neq 0 \dots 0$ **then** restart
    if $b_k = 0$ **then** return i

Algorithm 2 Sampling $D_{\mathbb{Z}^+,k\sigma_{\text{bin}}}$ for $k \in \mathbb{Z}$

Require: An integer $k \in \mathbb{Z}$ ($\sigma = k\sigma_{\text{bin}}$)
Ensure: An integer $z \in \mathbb{Z}^+$ according to D^+_{σ}
  sample $x \in \mathbb{Z}$ according to $D^+_{\sigma_{\text{bin}}}$
  sample $y \in \mathbb{Z}$ uniformly in $\{0, \dots, k-1\}$
  $z \leftarrow kx + y$
  sample $b \leftarrow \mathcal{B}_{\exp(-y(y+2kx)/(2\sigma^2))}$
  if $\neg b$ **then** restart
  return z

Algorithm 3 Sampling $D_{\mathbb{Z},k\sigma_{\text{bin}}}$ for $k \in \mathbb{Z}$

  Generate an integer $z \leftarrow D^+_{k\sigma_{\text{bin}}}$
  if $z = 0$ restart with probability $1/2$
  Generate a bit $b \leftarrow \mathcal{B}_{1/2}$ and return $(-1)^b z$

Fig. 1: The Bernoulli sampling technique for generating discrete Gaussian noise, as described by Ducas *et al.* [10].

$1.1001000010000001\dots$ is not constructed on-the-fly. Instead they use two 64-bit shift registers to store the expansion precomputed up to a precision of 128 bits, which outputs an $x \in D_{\sigma_{\text{bin}}}$. Uniformly random values $y \in \{0, 1, \dots, k-1\}$ are sampled which may require rejection sampling (for instance, the probability of a rejection for $k = 254$ is 2/256). The pipelined Bernoulli calculation stage takes a (y, x) tuple as input and computes $t = kx$ and outputs $z = t + y$ as well as $j = y(y + 2t)$. While z is retained in a register, the Bernoulli evaluation module evaluates the Bernoulli distribution of $b \leftarrow \mathcal{B}_{\exp(j/2\sigma^2)}$. If and only if $b = 1$ the value z is passed to the output, and discarded otherwise. The evaluation of $\mathcal{B}_{\exp(x/f)}$ requires independent evaluations of Bernoulli variables. Sampling from $\mathcal{B}_c$ is done by evaluating $s < c$ for a uniformly random $s \in [0, 1)$ and a precomputed c. The precomputed tables store values $c_i = \exp(2i/f)$, for $0 \leq i \leq l$, $f = 2\sigma^2$ where $l = \lceil \log_2(\max(j)) \rceil$, are stored in a distributed RAM. The $\mathcal{B}_{\exp(-x/f)}$ module then searches for one-bit positions u in j and evaluates the Bernoulli variable $\mathcal{B}_{c_u}$. They do this in a lazy manner; the evaluation aborts when the first bit has been found that differs between a random s and c. The technique saves randomness and runtime but incurs non-constant runtime. The chance of rejection is larger for the most significant bits, therefore these are scanned first to abort as quickly as possible. Lastly, a random one-bit is used to "sign" the samples and reject half of the samples where $z = 0$.

The Bernoulli sampler is suitable for hardware implementations as most operations work on single bits. However, due to the non-constant time behaviour of rejection sampling, buffers were introduced in [28] between each element to allow parallel execution and maximum utilisation of every component. This includes the distribution and buffering of random bits. To reduce the impact of buffering on resource consumption they included LIFO buffers that solely require a single port RAM and a counter, as the ordering of independent random elements does not need to be preserved by the buffer (as in the case with a FIFO). For maximum utilisation they evaluated optimal combinations of sub-modules and implemented two $\mathcal{B}_{\exp(-x/f)}$ modules fed by two instantiations of the Trivium stream cipher to generate pseudo-random bits. To date, no time-independent implementations of the Bernoulli sampler have been reported.

3.3 Cumulative Distribution (CDT) Sampling

The Cumulative Distribution Table (CDT) sampler requires a precomputed table of discrete Gaussian cumulative distribution function (CDF) values. The distribution symmetry is exploited to sample from $\mathbb{Z}^+$, saving 50% of required table storage space. The first and last samples of the table are kept 0 and 1 respectively, and a total of $N = \tau \times \sigma$ samples ($0 = S[0] < S[1] < \dots < S[N-3] = 1$) are required. Once the CDF values $S[\cdot]$ are computed, a sample r is drawn uniformly, $r \in [0, 1)$, with λ bits of precision, where the desired sample, x, is found satisfying interval $S[x] \leq r < S[x+1]$, occurring with probability $\rho[x] = S[x+1] - S[x]$. Initial table values (close to $x = 0$) are more probable than values near the end.

As the discrete Gaussian CDF is a sorted table, the binary search algorithm is used to find the position of the target value, (shown in Algorithm 4). The search space, comprising initially of the entire CDT (N samples), is dichotomously exhausted in every iteration of the algorithm. Pointers *min* and *cur* point to the first and the middle of the search space, respectively, while *jmp* maintains the number of search space samples reduced by half. In every iteration of the while loop, r is compared to the middle value (*cur*) of search spaces, whose upper or lower half is discarded depending on the comparison result. For a bounded interval, the number of comparisons required before a match is found does not exceed $\lceil \log_2(N) \rceil$. A uniformly sampled bit b is used to dictate the sign of result x.

The use of discrete Gaussian sampling based on large pre-computed CDTs was first proposed by Peikert [26], and adapted by Ducas *et al.* [10] for use in their signature scheme BLISS. Several reductions to the table size were

Algorithm 4 Sampling $D_{\mathbb{Z},\sigma}$ using the Cumulative Distribution Table (CDT) and binary searches

Require: Three Integers min, cur and jmp
  Discrete Gaussian CDT Samples such that, $N = \tau \times \sigma$, $0 = S[0] < S[1] < \dots < S[N-3] = 1$
  Sample a bit b uniformly in $\{0, 1\}$
  Sample r uniformly in $\{0, \dots, (2^\lambda - 1)\}$
Ensure: $min \leftarrow 0; cur \leftarrow (N/2); jmp \leftarrow cur;$
  while $(jmp > 0)$ **do**
    $cur \leftarrow min + jmp;$
    if $(r \geq S[cur])$ **then**
      $min \leftarrow cur;$
    $jmp \leftarrow jmp >> 1;$
  return $x = (-1)^b min$

suggested by Pöppelmann *et al.* [28] for a reconfigurable hardware implementation of BLISS. Firstly, the most significant m bits of r can be hashed to reduce the search space according to precomputed guide tables. Choosing $m = 8$ cuts the 2891 entries initially required for BLISS into 2^m intervals requiring guide tables holding the *min* and *cur* pointers. Thus, the complexity of the binary search is reduced from [11,12] searches to [1.3,1.7] searches. Secondly, the adoption of Lemma 1 (Section 2.3) on BLISS parameters significantly reduces standard deviation for CDT, accompanied by a table size reduction by a factor of 11. Thirdly, a floating point representation of CDT samples with a variable mantissa size skips leading zero storage, halving the table size.

These optimisations reduce the precomputed CDT sizes, consequently reducing search space and improving design throughput. However, hashing divides the search intervals into irregular sizes, meaning the binary search within intervals has non-constant bounds, making it susceptible to timing analysis attacks. The only CDT-based discrete Gaussian sampler promising a constant-time throughput for small standard deviations relies on a array of N parallel comparators, of λ-bits each, comparing against the uniform number r [29]. Each comparator returns a binary answer, the first comparator x satisfying $S[x] \leq r < S[x{+}1]$ is the required result. This fully pipelined design results in a single cycle per sample throughput but the large number of parallel comparisons renders it inefficient in terms of hardware resources. Du and Bai [9] optimise the hardware area by employing a piecewise comparison instead of a full λ-bit comparison in the binary search algorithm. To compare any λ-bit table sample and a uniform value of the same size, comparing the first m bits can give a definite result of being greater or smaller with probability $(2^m - 1)/2^m$ (99.6% for $m = 8$) and increases for larger m. In the case of an equality, m is increased and a larger comparison must be carried out. This lazy comparison scheme not only reduces the need for large comparators, but also economises the need for uniform sample bits required per output. Hashing is used to further narrow down the binary search time resulting in an area-efficient and yet high average throughput performance.

3.4 Discrete Ziggurat Sampling

Discrete Ziggurat sampling is adapted by Buchmann *et al.* [4] from an original method of continuous Ziggurat sampling, proposed by Marsaglia and Tsang [22]. It is an optimised form of rejection sampling, which divides the area under the curve into several rectangles (REC_{NUM}) of equal area, where the left hand side of each rectangle is aligned with the y-axis and the right hand side of each rectangle aligns with the curve of the discrete Gaussian distribution. This structure can be used to optimise rejection sampling of random points from a uniform distribution. The increase in the number of rectangles thus decreases the probability of rejection. Currently, there are no hardware implementations of this sampler. However, Buchmann *et al.* [4] propose a C++ implementation with several optimisations, depicted in Algorithm 5, and also a comparison of discrete Ziggurat with alternative sampling methods to show that this method is suitable for use when large standard deviations are required. In discrete Ziggurat sampling [4], firstly a random value, x, is generated and then x is checked for cases when x equals zero or lies comfortably within a given rectangle (i.e. $x \leq \lfloor x_{i-1} \rfloor$). Random samples including a random sign bit s are then generated. Otherwise, the sample is found to lie close to the curve and further calculations are required to establish whether x is above or below this curve (see lines 10 onwards in Algorithm 5).

Algorithm 5 Discrete Gaussian Sampling using Discrete Ziggurat [4]

Require: The number of rectangles, m, the lower right corners of rectangles $(\lfloor x_i \rfloor, \lfloor y_i \rfloor)$ for $i \in m$, precision λ;
1: while *true* do
2: Choose rectangle, $i \leftarrow \{1, ..., m\}$, choose sign bit $s \leftarrow \{0,1\}$, choose random bit $b \leftarrow \{0,1\}$ choose random value $x \leftarrow \{0, \ldots, \lfloor x_i \rfloor\}$, choose $y' \leftarrow \{0, \ldots, 2^\lambda - 1\}$
3: **if** $(0 < x \leq \lfloor x_{i-1} \rfloor)$ **then**
4:     return sx;
5: **else**
6:     **if** $x = 0$ **then**
7:         **if** $b = 0$ **then**
8:             return sx;
9:     **else**
10:         $\bar{y} = y' \cdot (\bar{y}_{i-1} - \bar{y}_i)$;
11:         **if** $\lfloor x_i \rfloor + 1 \leq \sigma$ **then**
12:             **if** $\bar{y} \leq 2^\lambda \cdot$ **sLine** $\vee\ \bar{y} \leq 2^\lambda \cdot (\bar{\rho}_\sigma(x_i) - \bar{y}_i)$ **then**
13:                 return sx;
14:         **else if** $\sigma \leq \lfloor x_{i-1} \rfloor$ **then**
15:             **if** $\bar{y} \geq 2^\lambda \cdot$ **sLine** $\vee\ \bar{y} > 2^\lambda \cdot (\bar{\rho}_\sigma(x) - \bar{y}_i)$ **then**
16:                 continue;
17:             **else**
18:                 return sx;
19:         **else**
20:             **if** $\bar{y} \leq 2^\lambda \cdot (\bar{\rho}_\sigma(x) - \bar{y}_i)$ **then**
21:                 return sx;
22:             **else**
23:                 reject sample;

Algorithm 6 sLine [4] for use in Algorithm 5

Require: $\lfloor x_{i-1} \rfloor, \lfloor x_i \rfloor, \bar{y}_{i-1}, \bar{y}_i, x$;
1: **if** $\lfloor x_i \rfloor = \lfloor x_{i-1} \rfloor$ **then**
2:     return -1
3:     Set $\hat{y}_i = \bar{y}_i$ and
4:     **if** $i = 1$ **then** $\hat{y}_{i-1} = 1$
5:     **else**$\hat{y}_{i-1} = \bar{y}_{i-1}$
6:     return $\frac{\hat{y}_i - \hat{y}_{i-1}}{\lfloor x_i \rfloor - \lfloor x_{i-1} \rfloor} \cdot (x - \lfloor x_i \rfloor)$

3.5 Knuth-Yao Sampling

Knuth and Yao [17] present a tree based algorithm for sampling from non-uniform distributions by using a minimal number of input uniform bits, close to the entropy of the probability distribution. Given a probability distribution represented by $p_0, p_1, \ldots, p_N$, while each sample is represented by a maximum of λ bits, the probability matrix P can be represented as a $N \times \lambda$ binary matrix (after adding leading/trailing zeros). A binary tree, called the discrete distribution generating (DDG) tree, can be constructed, with λ−levels, comprising of

two types of nodes: internal nodes (called I nodes) having two children each or terminal nodes (called T nodes) without children. The DDG is constructed so that the number of terminal nodes at the ith level of the tree equals the number of non-zero bits in the ith column of the probability matrix P. Each terminal node is marked by the row number of P. The sampling process is a random walk starting from the tree root moving down, consuming a single uniform bit at each step, taking the left node of the next level when encountering a 0 and right node otherwise. Hitting a terminal node outputs the integer label associated with it, generating a successful sample.

Figure 2 illustrates a DDG tree for a toy example. The tree levels equal the number of columns of P, while the number of terminal nodes match the non-zeros in P. A DDG tree can have at most $(N \times \lambda)$ terminal nodes and $(N \times \lambda) - 1$ internal vertices. Figure 2 shows equivalent tree and matrix representations of P, where each of the ith columns of the table has no more than $(2 \times i) \leq (N \times \lambda)$ entries, since that is the upper bound on the number of terminal nodes. The entire table does not need to be stored; instead P and the local information of one column suffices so that the information of the next column can be constructed at runtime.

Roy *et al.* [35] proposed a hardware friendly random walk for the discrete Gaussian sampler, based on the exploitation of ith column/tree level information to interpret the next. At any level i of the tree traversal, let d denote the distance between the visited node and the right most node of that tree level. At the DDG tree root (the 0th level), $d = 0$. Depending on a 0 or a 1 being encountered, the distance becomes $2 \times d + 1$ or $2 \times d$, respectively, after consuming one uniform bit. Next, they exploit the fact that in the DDG tree, all the I nodes are on the left and all the T nodes (that are non-zero and contribute to the Hamming weight of that particular M column) are on the right. Hence, subtracting the Hamming weight of the current column from the distance d at the ith level of the tree will show if the

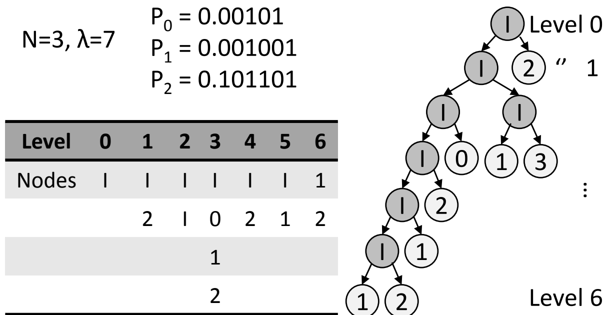

Fig. 2: The Knuth-Yao based discrete Gaussian sampler for a toy example: the probability matrix and the DDG tree (right) with its table based representation.

Algorithm 7 Sampling $D_{\mathbb{Z},\sigma}$ using Knuth-Yao algorithm from a given discrete Gaussian distribution

Require: Three Integers d, hit and ctr;
1: Discrete samples of Gaussian distribution as matrix P with $N \times \lambda$ dimension and $N = \tau \times \sigma$;
2: Sample bits uniformly in $\{0, 1\}$, store in array r;
3: Column-wise Hamming distance of P, i.e., $h_dist[j] = \sum_{i=0}^{N} P[i][j]$;
Ensure: $d \leftarrow 0; hit \leftarrow 0; ctr \leftarrow 0$;
4: **for** (int $col \leftarrow 0 : col < \lambda; col \leftarrow col + 1$) **do**
5: $d \leftarrow 2d + (!r[ctr++]) - h_dist[col]$
6: **if** $(d < 0)$ **then**
7: **for** (int $row \leftarrow 0 : row < N; row \leftarrow row + 1$) **do**
8: $d \leftarrow d + P[row][col]$;
9: **if** $(d == 0)$ **then**
10: $hit \leftarrow 1$;
11: **break;**
12: **if** (hit) **then**
13: **break;**
14: **return** $(-1)^{r[ctr++]}.row$:

current node is an I node or a T node, if the result is positive or negative respectively.

Algorithm 7 gives the procedural details of the Knuth-Yao sampling algorithm. Assertion of the `hit` signal indicates a successful sample and `ctr` is the iteration over the random binary samples array `r`. `h_dist` is a λ-element vector holding the column-wise Hamming distances of the matrix `P`. The loop at line 4 iterates over the columns, starting from the most significant bit position, and checks if a terminal node is hit in that level of the DDG tree, by checking the sign bit of `d`; each iteration consumes one uniformly sampled bit. In case a column is localised for a hit, the loop in line 7 iterates over all the rows in that column, until a `row` is localised where a `hit` is found, terminating the two loops. No uniform random bits are consumed during these iterations. A signed bit is attached to the sample, based on the uniformly sampled bit.

Devroye [8] shows that the required uniformly generated bits for a sample generation is at most two more than the entropy of the distribution. Hence, for the test case $\sigma_{\text{LP}} = 3.33$, the number of uniform bits required per sample is 5.3. The implementation requires minimal resources for storing the $N \times \lambda$ dimensional binary matrix `P`, a λ element `h_dist` vector of $\lceil \log_2(N) \rceil$ bits each, and several pointers for holding the `row`, `col`, and `d`, as well as a sampling termination signal `hit`.

Dwarakanath and Galbraith [11] suggest a compression of the probability matrix, as most of the distribution values close to the tail have an increasingly large number of leading zeros. A block variant of the Knuth-Yao algorithm was proposed that divides consecutive probabilities in different blocks with roughly the same number of leading zeros resulting in reasonable storage savings of P. Roy *et al.* [35] exploited this block based compression for the Knuth-Yao sampler in an FPGA implementation with $\sigma_{\text{LP}} = 3.33$. Instead of keeping a `h_dist` vector for column Hamming weight, they iterate over the columns of the `P` matrix, bit by bit. The iteration proceeds from bottom to top, consuming one cycle for each bit of the probability matrix. Since the discrete Gaussian values close to the centre are more likely, working from top to bottom instead significantly

accelerates the algorithm's average speed (as in Algorithm 7). Consequently, the implementation [35], though very lightweight, requires on average 17 clock cycles to generate one discrete Gaussian sample. Subsequent work by Roy *et al.* [34] improved the speed up to 2.5 cycles per sample, by table hashing in multiple stages. As these implementations were non-constant time, a random shuffle method is used to protect the discrete Gaussian distributed polynomial against timing analysis attacks, at the cost of additional FPGA slices. Clerq *et al.* [7] present a software implementation of ring-LWE encryption on a 32-bit ARM Cortex-M4F microcontroller using a Knuth-Yao based fast discrete Gaussian sampler, bettering all existing encryption implementations in software, where discrete Gaussian sampling requires an average 28.5 cycles per sample.

4 Timings Attack & Design Goals

Timing attacks were proposed for the first time by Kocher [18]. These attacks exploit the time difference between operations to gain information about the secret key. Exploitable timing differences can be caused by routines whose execution time depends on the secret data or by the difference in data access pattern caused by memory hierarchy. These attacks have been successfully applied against several cryptographic schema, including lattice-based schemes (in particular NTRU [38]). Thus in the context of LBC, it is crucial to implement schemes resistant against timing attacks.

The need for a discrete Gaussian sampler resistant against *timing attacks* is emphasised by the NIST call for implementations of encryption and signature schemes (of which discrete Gaussian samplers are core components) explicitly requiring resistance against side-channel attacks [25]. Software implementation of Gaussian samplers have been shown to leak information through the timing channel [36]. Bruinderink *et al.*, in particular, presented a time side-channel attack, exploiting the cache patterns [3], on the CDT sampler. The majority of hardware designs reported so far [35], [28], [15] are susceptible to timing attacks. Resistance against timing attacks is achieved when all the operations involving secret information are executed in a time which is independent from the secret data.

Time independence is a property which implies that the computation time of an algorithm does not depend on the input data. As a consequence, when the computed data involve secret information, a time independent computation guarantees that no information about it is leaked via the timing channel. This property can be achieved in several ways, the most common methods are ensuring **constant execution time** [30] or random **shuffling** [34] of the secret values.

The discrete Gaussian sampler proposed by Pöppelmann and Güneysu [30], which targets constant execution time, implements a time-independent CDT sampler for encryption. The area overhead of that design is however too high to be practical. Roy *et al.* [34] also presented a discrete Gaussian sampler resistant against timing attacks. The design implements a fast Knuth-Yao based *non-constant time* Gaussian sampler which generates a batch of samples, which is subsequently *shuffled* to disassociate the related timing information.

Time independence is not the only desired feature: implementations of discrete Gaussian samplers should also ensure fast response time (for high throughput) and require minimal area occupation (for constrained devices). In this paper these design goals are considered and the limitations of previous work are overcome by proposing *independent-time* and *practical* implementations of all four discrete Gaussian sampling schemes currently used in LBC. However, throughput and area are considered as secondary variables to be optimised, since the main objective is to ensure time-independence of the proposed designs.

5 Gaussian Sampler Architectures

5.1 Constant-Time Bernoulli Sampling

Discrete Gaussian sampling using Bernoulli is performed in constant-time when the table comparison (of values c_i, seen in Section 3.2) is always completely evaluated. Thus a comparison would not be aborted if one mismatch between the randomly generated bit r and the table value c is found. However, if a rejection of the Bernoulli variable leads to the rejection of a complete sample, is $r \geq c$ an abort can be tolerated. For acceptance $(r < c)$ instead, the full comparison has to be made. The dependency on the input value of x would in fact leak information. The leakage can be mitigated by evaluating the whole table in a pipelined fashion, so that no secret addresses are used (the whole table has to be read). This however requires a large number of random bits, generated using Trivium x32 as a PRNG. The PRNG produces 32 random bits, sufficient for the whole table comparison, and they are stored in a register, ready to be used when $j = y(y + 2kx)$ is calculated. In the case of a rejected value, the evaluation aborts early, since rejection does not leak any secret information.

Figure 3 illustrates the high-level architecture of the proposed constant-time Bernoulli sampler. The binary Gaussian module includes the reduced precision $\lambda = 64$-bits and uses a 64-bit shift register (implemented in a LUTM on the target FPGA) to store the precomputed expansion. This operation, as well as the uniformly random value $y \in \{0, 1, \ldots, k-1\}$, are also improved by incorporating an unrolled x8 Trivium as a PRNG. Previous designs [28] required 12-bits of randomness for the binary Gaussian component and 8-bits of randomness for the uniform value k. This proposed design, which combines a x8 unrolled Trivium with a reduced precision allows the computation in two clock cycles instead of ≈ 20 clock cycles using a single bit per clock PRNG. The values of j and z are used in the evaluation, j is used to evaluate $b \leftarrow \mathcal{B}_{\exp(-j/2\sigma^2)}$ where if $b = 1$ the value z is accepted and output, otherwise it is rejected.

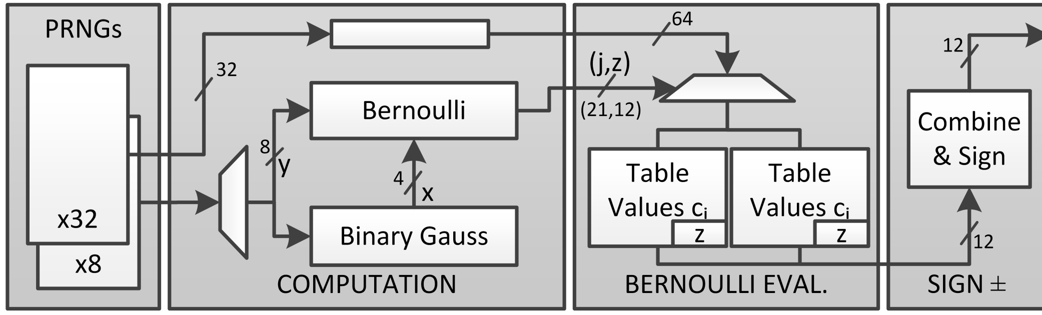

Fig. 3: High level architecture of the proposed constant-time Bernoulli samplers with variables and their bit lengths given, which uses a x8 and x32 Trivium as a PRNG.

Since Gaussian convolutions are integrated (Section 2.3), employing two table evaluation components decreases the clock cycle count per sample. Each evaluation is able to compute a 128-bit comparison on every clock cycle, meaning per clock cycle two rows can be compared. Once the pre-calculation components are completed and the pipeline is full, input values to the tables will be ready after every clock cycle, meaning a discrete Gaussian sample takes exactly $\lceil\lceil\log_2(\max(j))\rceil/2\rceil + 2$ cycles, which includes $\lceil\lceil\log_2(\max(j))\rceil/2\rceil$ for $x_1 \leftarrow D_{\sigma'}$, $+1$ more clock cycle for $x_2 \leftarrow D_{\sigma'}$, and $+1$ more clock cycle for them to be combined as $x = x_1 + 11x_2 \leftarrow D_{\sigma}$.

The sampler is designed generically such that it can operate for both standard deviations, $\sigma_{\text{LP}} = 3.39$ and $\sigma_{\text{BLISS}} = 215.73$, by adapting the lookup table and the constants. Precomputed tables are stored as distributed RAM, and calculated as $\lambda \log_2 2.4\tau\sigma^2$ which for $\sigma_{\text{BLISS}} = 215.73$ requires 784 bits and improves on [28] by $\approx 48\%$ in table space, and for $\sigma_{\text{LP}} = 3.39$ requires 400 bits. Furthermore, the proposed Bernoulli sampler generates a Gaussian sample in 13 clock cycles for BLISS signature parameters and 7 clock cycles for LP encryption parameters, which as well as being the first Bernoulli sampler to operate in constant-time, also betters the speed of the previous hardware implementations of [28], and competes well with [15].

5.2 Constant-Time CDT Sampling

The binary search for a desired sample can result in an early termination, when the equality comparison of uniformly sampled r and the $S[cur]$ results in an exact match, though this early termination is only likely with a very small probability. Algorithm 4 for CDT sampling qualifies the inequality (instead of 64-bit equality comparison), hence avoiding early termination bounding the algorithm for $\left[\lfloor\log_2(N)\rfloor, \lceil\log_2(N)\rceil\right]$ search iterations before generating a result every time. For CDT where N is a power of two, the algorithm performs inherently in constant-time; for all other N, the algorithm is tweaked to occasionally perform an extra iteration to ensure the algorithm complexity is fixed to $\lceil\log_2(N)\rceil$.

For the generation of discrete Gaussian samples with $\sigma_{\text{LP}} = 3.33$, the CDF table S consists of $N = 32$ ($\lceil\tau \times \sigma_{\text{LP}}\rceil$) entries, each with $\lambda = 64$ bits of precision. Hence, a single ported BRAM, with a 5-bit address and 64-bit data ports suffices for the design. A $64\times$ unrolled Trivium is used as a PRNG for the generation of the uniform samples r, where the initialisation of the module is handled externally at the startup and controlled by the binary search state machine (referred to as `BinSearch`). The Trivium module is activated by the `enable` output pulse signal from `BinSearch`, so the uniform samples are only generated when required, saving circuit power. For $\sigma'_{\text{BLISS}} = 19.47$, the CDT table S is larger ($N = 184$ holding $N \times \lambda \approx 11$K bits). According to convolution Lemma 1, for each valid discrete Gaussian sample, two samples of $\sigma'_{\text{BLISS}} = 19.47$ are required. A single `BinSearch` state-machine instance would take $2 \times \lceil\log_2(184)\rceil = 16$ cycles for generation of a single valid sample. However, a better throughput model uses two instances of state-machine to parallelise two independent searches (seen in Figure 4). The two state machines `BinSearch0` and `BinSearch1` each get a 64-bit uniformly random number from the Trivium-based PRNGs, in two consecutive clock cycles. The `BinSearch` state-machines initiate their processing from the `START` state, resetting three pointers, namely, min, cur, and jmp, with initial values, as given in Algorithm 4. They transition unconditionally to the `SEARCH` state in the next clock cycle that updates their three pointers as per the result from their 64-bit comparison operation. After exactly 8 cycles an appropriate x is found, and the state generates a single bit `hit` to signal this occurrence. This hit also activates the Trivium module to request a new uniformly random 64-bit value. `BinSearch0` and `BinSearch1` share the same dual ported BRAM, each port having 8-bit address and 64-bit data ports. The state machines work independently to generate two random samples $x_1, x_2 \leftarrow D_{\sigma'}$ in 8 clock cycles (shown in Figure 4); the two samples are buffered on their respective *hit* signals into registers. The samples are then combined as $x = x_1 + 11x_2$ where a sign bit is also attached.

5.3 Time-Independent Knuth-Yao Sampling

Inherently, the Knuth-Yao sampler operates in non-constant time. Considering the column and row local-

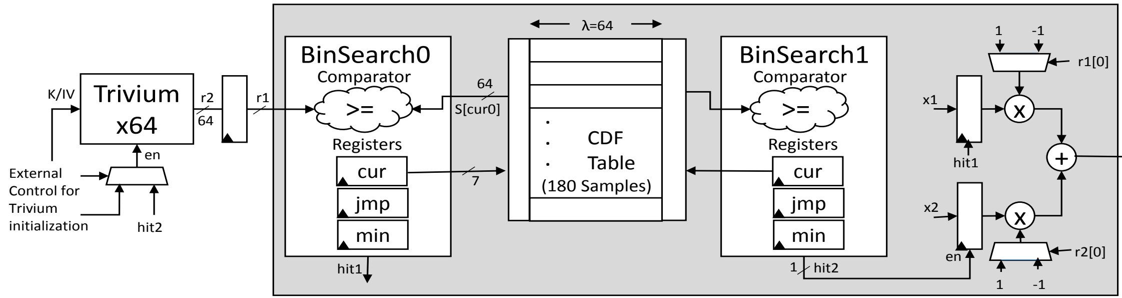

Fig. 4: The CDT discrete Gaussian sampler for $\sigma_{\text{BLISS}} = 215$, using two `BinSearch` state machines each accessing the CDF table for $\sigma'_{\text{BLISS}} = 19.47$.

isation phases take one clock cycle to jump from one column/row to the other, successful sampling requires at least 2 cycles if the 0th row and 0th column are localised for a hit, with the maximum being $(\lambda \times N)$ when the last row and last column get a hit. The average response time is ≈ 10.6 cycles since the column and row localisation each require ≈ 5.3 cycles.

Figure 5 gives an architectural description of the proposed Knuth-Yao sampler in hardware, given a probability matrix P. Trivium is used as a PRNG for the generation of the uniformly sampled bit `r` that is inverted to get `r_n`. The initialisation of Trivium is handled externally at startup, which afterwards is controlled via the `KYSearch` state-machine. An `en` signal is used to activate the PRNG to produce randomness only when required. The `KYSearch` state-machine initiates its processing from the `START` state, resetting output registers, `row`, `col`, `d`, `en` and `hit` with initial values, as given in Algorithm 7. It transitions unconditionally to the `SEARCH` state that updates the output registers as per the state of the d pointer. As long as d is a non-zero positive number, the state-machine keeps changing columns, consuming random bits and updating d accordingly. If `d` is a negative number, then the column for a hit is already localised and the row search starts without changing column values. Since no new random bits are required, `en` is kept low and `d` is updated by adding the `P` matrix values until `d` reduces to 0. Assertion of a `hit` ends the sampling process, where the row number is assigned a sign bit based on a uniformly sampled bit.

To ensure ease of access, the P matrix transpose is stored as a distributed ROM or in a BRAM in the FPGA implementation. When using a BRAM, storing a mere 64×32-bit P matrix is excessive. For better resource utilisation, the h_dist vector of Hamming distances is also stored in the same BRAM. Additionally, hashing is employed to boost the throughput, since the response time is non-constant and the same BRAM holding the P matrix can also store the hash tables. To consider 8 bits of uniform random numbers at a time, a hash table with 256 entries is easily accommodated in BRAM, for the case where $\sigma_{\text{LP}} = 3.33$, 249 out of these 256 entries result in a hit (97.26%). Otherwise, the hash table entries keep the `d` value to be loaded in the `KYSearch` state-machine directly and scanning of the first 8 columns is skipped, improving throughput.

To make the sampler independent-time, operations could be made constant-time, that is the largest possible cycle count $(\lambda \times N)$ must be ensured before a discrete Gaussian sample is generated as an output. For $\sigma_{\text{LP}} = 3.33$, it turns out to be too slow to be practical (2000 cycles per valid sample generation, and worse for signature parameters). Hence, to achieve timing independence, the discrete Gaussian samples are instead shuffled after the generation of a complete block. Considering a block size of n samples, if n_1 are generated by hashing then the remaining $n_2 = n - n_1$ are generated by the `KYSearch` state-machine. The samples are stored in another BRAM such that n_1 samples are accommodated at the top of the BRAM with the remaining at the bottom. A simplistic shuffling algorithm is implemented, the Fisher-Yates shuffle [12] (also used in [34]), which goes over each of the n_2 samples, swapping these within randomly generated locations $[0, n_1 - 1]$. A simple `shuffler` state-machine enables each swap in two clock cycles considering a dual-ported BRAM. Due to the small percentage of n_2 samples in BRAM (2.7%), shuffling does not significantly exacerbate the sample generation throughput. However, the state-machine and the BRAM require extra FPGA resources.

The Knuth-Yao implementation relates the number of random bits required (and consequently the running time of the algorithm) to the distribution entropy (or standard deviation). For $\sigma_{\text{BLISS}} = 215$, even after using Lemma 1, the running time will be more than 20 clock cycles per sample, which is far too slow for practical purposes. Hashing would also require much larger tables to be effective. Consequently, no further work is undertaken on the Knuth-Yao sampler for signature parameters.

5.4 Constant-Time Discrete Ziggurat Sampling

To date, there are no existing hardware designs of the discrete Ziggurat Gaussian sampler. There are several possible reasons for this: firstly, the discrete Ziggurat sampling algorithm requires several costly computations, secondly, the generation of suitable rectangles for sampling is non-trivial and can require multiple iterations, and thirdly,

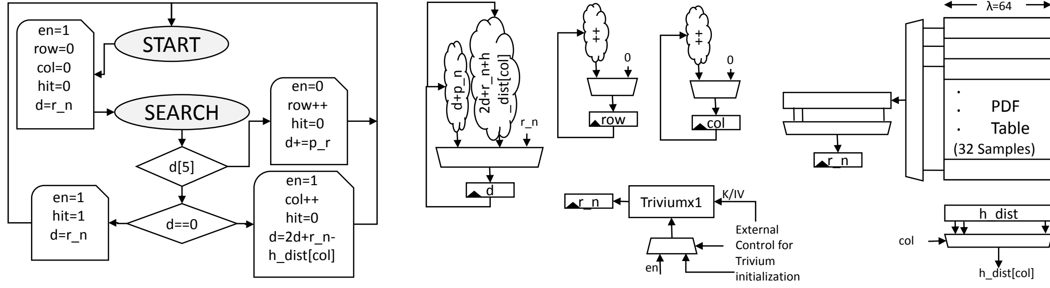

Fig. 5: A Knuth-Yao based discrete Gaussian sampler for $\sigma_{\text{LP}} = 3.33$

there are several alternative sampling algorithms which are better suited to the hardware platform.

In this research, the first hardware design of a discrete Gaussian Ziggurat sampler on FPGA is presented. To adapt the algorithm to suit the hardware platform, the rectangle generation and several pre-computations are completed offline. This is a reasonable assumption to make, as previously mentioned in other sampling algorithms (such as the precomputed lookup tables used in Bernoulli and CDT samplers). The number of rectangles (REC_{NUM}) used in the proposed design is set to eight. Such parameters would benefit from increased flexibility or on-the-fly calculation in future implementations. Parameters are stored as binary vectors, and fixed point precision (λ bit) is used to minimise memory consumption on the FPGA. Two's complement representation is used for the binary sample values, where the most significant bit represents the sign of the given sample. The floored x coordinates and λ bit precision y coordinates of the rectangles and a table of precomputed probability values (that is, $(\rho_\sigma(x_i) - \bar{y}_i) * 2^\lambda$) are stored on the FPGA, called `STORE X`, `STORE Y`, and `STORE RHO` respectively. Additionally, `SLINE COMP` stores the precomputed fractional division required in the sLine operation. These precomputations enable efficient hardware implementation at the cost of flexibility. For this reason, software implementations of Ziggurat are preferred to any hardware implementation.

The design follows Algorithm 5, where a series of comparisons are used to generate discrete Gaussian samples. Figure 6 depicts the proposed hardware architecture at a high level. An unrolled Trivium x8 is employed and a buffer is used to store intermediate values. A counter controlled by a state-machine is used to ensure the buffer is full before initial sample generation. The design require n uniformly random bits per sample, where $n = 2 + \lambda + (2 \times \text{REC}_{\text{NUM}})$ given the number of rectangles REC_{NUM}. For sampling, a random integer value x and a rectangle value i are randomly chosen using Trivium outputs. If x is less than the floored x coordinate of i, x_i is stored in `STORE X`, it is output along with a sign bit, s. Otherwise, several other comparisons are carried out to verify if x falls under the discrete Gaussian curve. In this case, a y_{dash} value is uniformly generated with λ-bits of precision, and subsequently used to generate $\bar{y}$, such that $\bar{y} = y_{dash} \cdot (\bar{y_{i-1}} - \bar{y_i})$. Note that $\bar{y_i}$ are stored in `STORE Y`. Then, $\bar{y}$ is used within three inequalities to decide whether to reject the sample x, that is, to decide in which part of the rectangle the randomly generated integer x lies (as seen in Algorithm 5). For further details see the original proposed algorithm and software design by Buchmann *et al.* [4].

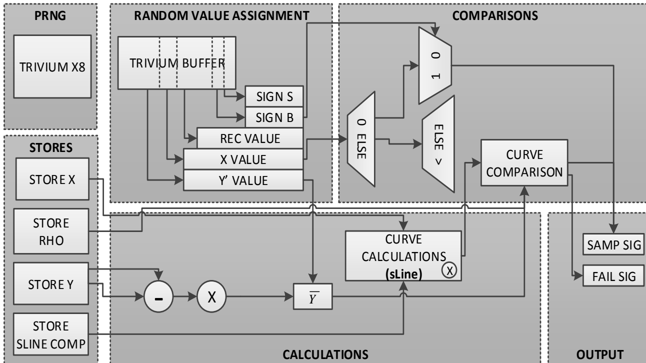

Fig. 6: High level hardware architecture of discrete Ziggurat sampler for $\sigma_{\text{LP}} = 3.33$

For the signature parameters, the same design is used twice and the sampling process is repeated to generate two samples, and combined in the same way as in the other sampling techniques. To make the sampler perform in constant-time, the state-machine controls signals and ensures all comparisons are carried out, regardless of success or failure, within a set number of clock cycles. This design choice brings a small cost of additional clock cycles per sample generated.

6 Comparison & Results

In this section the performance results are described and compared against the same implementations for encryption (for $\sigma_{\text{LP}} = 3.33$) and signatures (for $\sigma_{\text{BLISS}} = 215$). The designs are implemented on either the Spartan-6 LX25-3, Virtex-5 LX30, or Virtex-6 LX75 FPGA devices to compare with previous work, where the results obtained are all post place-and-route (PAR) using Xilinx ISE 14.7. Throughput and throughput per area are evaluated for all schemes in terms of sampling operations per second (Ops/s) and sampling operations per second per slice of FPGA (Ops/s/S), respectively, in Tables 2-5. Significantly more random bits are required to guarantee constant computation time. Random bits are produced by Triviums x8 and x32. The resource needed for these

–when compared with the standard Trivium x1 used in Section 5.3 – are rather negligible: 28 additional LUTs, 63 additional flip-flops, and 15 additional slices for Trivium x8 and 26 additional LUTs, 147 additional flip-flops, and 21 additional slices for Trivium x32.

6.1 Bernoulli Results

Table 2 shows the performance of the constant-time Bernoulli samplers in hardware. The results are compared to other implementations which target lattice-based encryption and signature schemes but without protection against timing attacks. To achieve constant computation time, the proposed Bernoulli samplers need to perform many more comparisons than the previously proposed designs [28], [15]. For this reason, the number of flip-flops in this design is larger than others. However, the needed comparisons can be performed in a single clock cycle, where instead LUT and slice usage has decreased. This has mainly been achieved by incorporating x8 and x32 unrolled Trivium components, which simplify the overall design and alleviate the need for excess data buffers. The halving of the precision parameter also contributed to the reduction of the LUT and slice usage. It can be seen from Table 2 that, despite the increase in flip-flop consumption, the Bernoulli samplers for both standard deviations improve on previous designs.

6.2 Knuth-Yao Results

Table 3 compares the proposed Knuth-Yao samplers for encryption parameters with the state-of-the-art. Roy *et al.* [35] present a discrete Gaussian sampler design requiring 17 clock cycles per sample. The slow throughput was primarily due to the bit-by-bit scanning of the probability matrix `P`. Multiple bit scanning per clock after column localisation does slightly improve throughput, with an additional resource cost. Further, Roy *et al.* [34] present low latency variants of the same architecture changing the dimensions of the matrix `P`, each requiring 17 clock cycles per sample. Multi-stage table hashing improves throughput up to ≈ 2.5 clock cycles per sample and subsequently a shuffler makes the sampler time independent. However, hardware implementation results for the shuffler are not provided [34].

The first implementation (without BRAMs and hashing) compares to the existing work by [34] (without hashing). In the second implementation, hashing is undertaken; the BRAM is used to store the P matrix, the h_dist and the hashing table. At the cost of one additional BRAM, this implementation improves on the Ops/s/S of the best existing Knuth-Yao implementation by a factor of 2. The last implementation also includes a shuffler for timing independence; hence another BRAM for generated discrete Gaussian samples is added and the slice budget increases to accommodate the shuffler state machine. A consequent decrease in Ops/s/S follows.

6.3 Cumulative Distribution Table Results

For the CDT sampler, FPGA implementations with and without BRAMs are proposed. For $\sigma_{\text{LP}} = 3.33$, a single port distributed ROM is used. The number of slices can be significantly reduced if BRAMs are utilised, as shown in Table 4, where one instance of `RAMB36` is used in a 64×32 configuration. However, in this configuration the maximum operable frequency is halved. The reduction in slices is higher for $\sigma_{\text{BLISS}} = 215$ due to the larger distribution table being shifted to BRAM.

Table 4 compares the proposed CDT samplers with state-of-the-art. The only other constant-time CDT implementation for $\sigma_{\text{LP}} = 3.33$, proposed by Pöppelmann and Güneysu [30], operates at a frequency of more than $4\times$ lower. The slice count is also 5x larger, contributed primarily by as many parallel comparators as the S table words. Hence, despite the reported CDT implementation generating a single sample per cycle [30], the design in this research proves to be a very lightweight, constant-time alternative, outperforming it by a factor of around $5\times$ in terms of Ops/s/S. The CDT sampler by Du and Bai [9] is lightweight and achieves good average throughput. However, their proposed design is not an independent time implementation. For $\sigma_{\text{BLISS}} = 215$, the implementation by Pöppelmann *et al.* [28] also operates in non-constant time, is costlier in terms of slice consumption, and slower in terms of throughput per slice. Since their reported design consumes BRAM, when compared to this implementation (with BRAM), their design remains $\approx 5\times$ inferior in terms of Ops/s/S. However it requires around $3\times$ less uniform random bits per sample, compared to the proposed constant-time designs.

6.4 Discrete Ziggurat Results

Table 5 shows the performance of the proposed constant-time discrete Ziggurat. There are no previous hardware designs of this sampler; thus the discrete Ziggurat sampler can be compared with the other samplers (Tables 2-4). The discrete Ziggurat sampler design requires a large amount of area resources, and achieves a relatively slow performance compared to the other samplers. For these reasons, hardware discrete Ziggurat samplers are not recommended and are not included in Figure 7.

7 Recommendations and Conclusion

Figure 7 plots the post-PAR results for CDT, Knuth-Yao (KY) and Bernoulli (Ber) samplers, with and without RAMs, targeting the same FPGA device. Results for the discrete Ziggurat samplers are not included, due to their inefficiency. For encryption, the RAM-free CDT (CDT_Enc) sampler surpasses all others in terms of an overall balanced performance with area, throughput, and timing independence. If the use of additional BRAMs is considered, the Knuth-Yao time-independent implementation (KY_Enc_RAM) has the best overall performance in terms of low area, high throughput, and also the

TABLE 2: Post-place and route results of the Bernoulli sampler for encryption ($\sigma_{\text{LP}} = 3.39$) and signatures ($\sigma_{\text{BLISS}} = 215.73$), in comparison to those targeting the same discrete Gaussian parameters with non-constant operating time.

Op.	Implementation	Device	λ	LUT/FF /Slice	BRAM /DSP	Freq. (MHz)	Clock Cycles	Rand. Bits	Ops/s ($\times 10^6$)	Ops/s/S ($\times 10^6$/S)	Time Ind.
$\sigma_{\text{LP}} = 3.33$	This work	XC6SLX25-3	64	679/1580/279	0/0	133	7	85	19	0.06	✓
				516/1475/201	2/0	167	7	85	23.86	0.12	✓
	Howe *et al.* [15]	XC6SLX25-3	64	846/934/297	0/0	129	≈ 12	≈ 26	10.75	0.03	✗
	Pöppelmann *et al.* [31]	XC6SLX9-2	80	132/40/37	0/0	136	≈ 144	≈ 96	0.944	0.02	✗
$\sigma_{\text{BLISS}} = 215.73$	This work	XC6SLX25-3	64	1001/1842/356	0/0	139	13	85	10.69	0.03	✓
				571/1480/167	3/0	171	13	85	13.15	0.08	✓
	Pöppelmann *et al.* [28]	XC6SLX25-3	128	1231/1134/452	0/1	137	≈ 17.95	≈ 33	7.63	0.01	✗

TABLE 3: Post-place and route results of the Knuth-Yao sampler for encryption ($\sigma_{\text{LP}} = 3.33$), in comparison to existing work with same discrete Gaussian parameters.

Op.	Implementation	Device	λ	LUT/FF /Slice	BRAM /DSP	Freq. (MHz)	Clock Cycles	Rand. Bits	Ops/s ($\times 10^6$)	Ops/s/S ($\times 10^6$/S)	Time Ind.
$\sigma_{\text{LP}} = 3.33$	Roy *et al.* [35]	5VLX30	90	140/66/47	0/0	333	≈17	≈5.3	19.61	0.42	✗
				149/69/53	0/0	303	≈16	≈5.3	18.94	0.36	✗
	Roy *et al.* [34]	5VLX30-3	90	101/81/38	0/0	344	≈17	≈5.3	20.28	0.53	✗
				105/60/32	0/0	400	≈17	≈5.3	23.53	0.74	✗
				102/48/30	0/0	384	≈17	≈5.3	22.62	0.75	✗
				118/48/35	0/0	333	≈13	≈5.3	133.33	3.81	✗
	This work	5VLX30-3	64	99/21/35	0/0	310	≈10	≈5.3	31.02	0.89	✗
				59/25/22	1/0	212	≈1.16	≈8.3	183.02	8.32	✗
				133/52/48	2/0	212	≈1.23	≈8.3	172.60	3.60	✓

lowest number of bits required per sample. For signatures, the RAM-free CDT implementation (CDT_Sign) proves to be an overall winner, followed by the Bernoulli sampler (Ber_Sign), being around 2x more expensive in terms of slices. In conclusion, this research provides a thorough investigation of the practical discrete Gaussian samplers (CDT, Knuth-Yao, Bernoulli, and discrete Ziggurat) used in lattice-based cryptosystems. Novel time-independent implementations are presented, ensuring resistance against timing attacks. Moreover, the proposed hardware sampler designs clearly outperform most of those previously proposed.

REFERENCES

[1] M. Ajtai, "Generating hard instances of lattice problems (extended abstract)," in *STOC*, 1996, pp. 99–108.

[2] E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe, "Post-quantum key exchange - a new hope," Cryptology ePrint Archive, Report 2015/1092, 2015, http://eprint.iacr.org/.

[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," Cryptology ePrint Archive, Report 2016/300, 2016.

[4] 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 *SAC*, 2013, pp. 402–417.

[5] CESG, "Quantum key distribution: A CESG white paper," February 2016. [Online]. Available: https://www.cesg.gov.uk/white-papers/quantum-key-distribution

[6] CNSS, "Use of public standards for the secure sharing of information among national security systems," Committee on National Security Systems: CNSS Advisory Memorandum, Information Assurance 02-15, July 2015.

[7] R. De Clercq, S. S. Roy, F. Vercauteren, and I. Verbauwhede, "Efficient software implementation of ring-LWE encryption," in *DATE*. EDA Consortium, 2015, pp. 339–344.

[8] L. Devroye, "Sample-based non-uniform random variate generation," in *WSC*. ACM, 1986, pp. 260–265.

[9] C. Du and G. Bai, "Towards efficient discrete Gaussian sampling for lattice-based cryptography," in *FPL '15*. IEEE, 2015, pp. 1–6.

[10] L. Ducas, A. Durmus, T. Lepoint, and V. Lyubashevsky, "Lattice signatures and bimodal Gaussians," in *CRYPTO (1)*, 2013, pp. 40–56, full version: https://eprint.iacr.org/2013/383.pdf.

[11] N. C. Dwarakanath and S. D. Galbraith, "Sampling from discrete Gaussians for lattice-based cryptography on a constrained device." *Appl. Algebra Eng. Commun. Comput.*, pp. 159–180, 2014.

[12] R. A. Fisher and F. Yates, "Statistical tables for biological, agricultural and medical research." *Statistical tables for biological, agricultural and medical research.*, pp. 25–27, 1948, third Ed.

[13] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *STOC*. ACM, 1996, pp. 212–219.

[14] ——, "Quantum mechanics helps in searching for a needle in a haystack," *Physical review letters*, vol. 79, no. 2, p. 325, 1997.

[15] J. Howe, C. Moore, M. O'Neill, F. Regazzoni, T. Güneysu, and

TABLE 4: Post-place and route results of the Cumulative Distribution Table (CDT) sampler for encryption ($\sigma_{\text{LP}} = 3.33$) and signatures ($\sigma_{\text{BLISS}} = 215$), in comparison to existing results with same discrete Gaussian parameters.

Op.	Implementation	Device	λ	LUT/FF /Slice	BRAM /DSP	Freq. (MHz)	Clock Cycles	Rand. Bits	Ops/s ($\times 10^6$)	Ops/s/S ($\times 10^6$/S)	Time Ind.
$\sigma_{\text{LP}} = 3.33$	Pöppelmann & Güneysu [30]	6VLX75T-2	80	863/6/231	0/0	61	1	85	61	0.26	✓
			100	1157/6/314	0/0	58	1	85	58	0.18	✓
	This work	6VLX75T-2	64	112/19/43	0/0	297	5	64	59.4	1.38	✓
				53/17/15	1/0	193	5	64	38.6	2.57	✓
	Du & Bai [9]	5VLX30	90	43/33/17	1/0	259	≈2.28	≈9.44	113.6	6.68	✗
				85/65/39	1/0	256	≈1.14	≈9.44	224.6	5.76	✗
$\sigma_{\text{BLISS}} = 215$	Pöppelmann *et al.* [28]	6SLX25-3	128	928/1121/299	1/0	129	≈7.5	≈21	17.2	0.06	✗
	This work	6SLX25-3	64	577/64/179	0/0	130	8	64	16.3	0.09	✓
				130/48/44	2/0	126	8	64	15.8	0.36	✓

TABLE 5: Post-place and route results of the discrete Ziggurat sampler for encryption ($\sigma_{\text{LP}} = 3.33$) and signatures ($\sigma_{\text{BLISS}} = 215$)

Op.	Implementation	Device	λ	LUT/FF /Slice	BRAM /DSP	Freq. (MHz)	Clock Cycles	Rand. Bits	Ops/s ($\times 10^6$)	Ops/s/S ($\times 10^6$/S)	Time Ind.
$\sigma_{\text{LP}} = 3.33$	This work	XC6SLX25-3	64	563/785/785	0/26	60.3	9	82	6.7	0.008	✓
$\sigma_{\text{BLISS}} = 215.73$	This work	XC6SLX25-3	64	526/791/800	0/27	59.9	10	164	5.99	0.007	✓

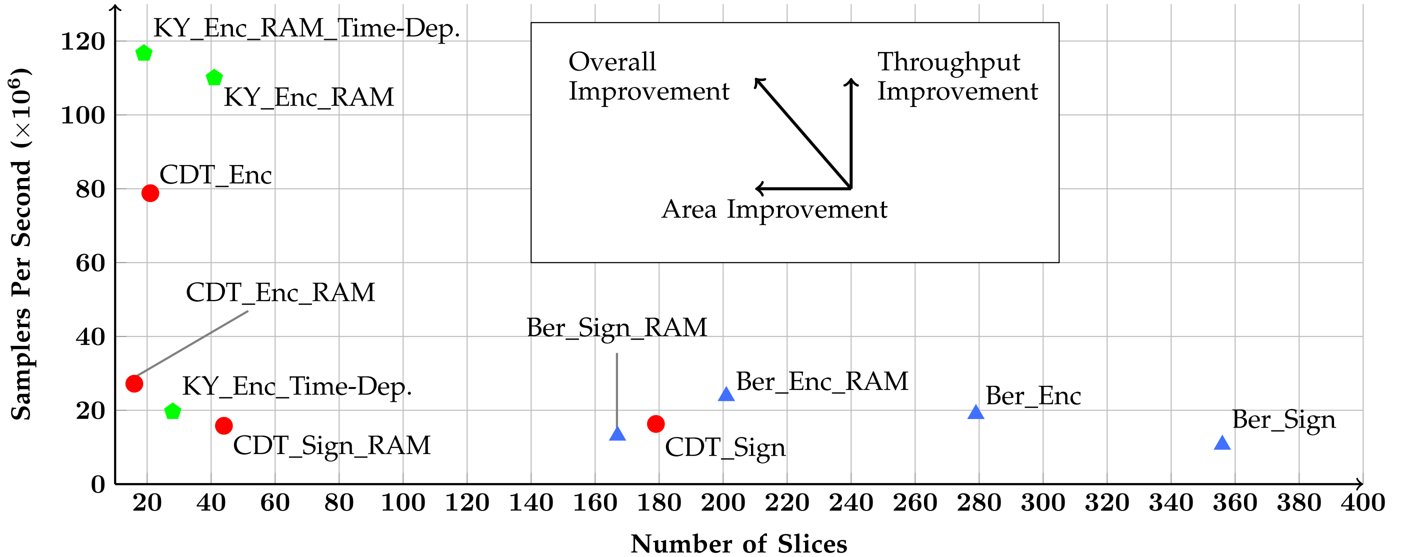

Fig. 7: Graphical performance results of the proposed discrete Gaussian samplers, on the Spartan-6 LX25-3 FPGA, with and without RAM use. All results are time-independent unless otherwise stated (Time-Dep.).

K. Beeden, "Lattice-based Encryption Over Standard Lattices in Hardware," in *DAC '16*. ACM, 2016, pp. 162:1–162:6. [Online]. Available: http://doi.acm.org/10.1145/2897937.2898037

[16] J. Howe, T. Pöppelmann, M. O'Neill, E. O'Sullivan, and T. Güneysu, "Practical lattice-based digital signature schemes," *TECS*, vol. 14, no. 3, p. 41, 2015.

[17] D. E. Knuth and A. C. Yao, "The complexity of nonuniform random number generation," *Algorithms and complexity: new directions and recent results*, pp. 357–428, 1976.

[18] P. C. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," in *CRYPTO*. Springer, 1996, pp. 104–113.

[19] R. Lindner and C. Peikert, "Better key sizes (and attacks) for LWE-based encryption," in *CT-RSA*, 2011, pp. 319–339.

[20] V. Lyubashevsky and D. Micciancio, "Generalized compact knapsacks are collision resistant," in *ICALP*. Springer, 2006, pp. 144–155.

[21] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and

learning with errors over rings," in *EUROCRYPT*, 2010, pp. 1–23.

[22] G. Marsaglia, W. W. Tsang *et al.*, "The Ziggurat method for generating random variables," *Journal of statistical software*, vol. 5, no. 8, pp. 1–7, 2000.

[23] D. Micciancio, "Generalized compact knapsacks, cyclic lattices, and efficient one-way functions," *Comput. Complex.*, vol. 16, no. 4, pp. 365–411, Dec. 2007.

[24] D. Micciancio and C. Peikert, "Hardness of SIS and LWE with small parameters," in *CRYPTO (1)*, 2013, pp. 21–39.

[25] D. Moody, "Post-quantum cryptography: NIST's plan for the future," Talk given at PQCrypto '16 Conference, 23-26 February 2016, Fukuoka, Japan, February 2016. [Online]. Available: https://pqcrypto2016.jp/data/pqc2016_nist_announcement.pdf

[26] C. Peikert, "An efficient and parallel Gaussian sampler for lattices," in *CRYPTO*, 2010, pp. 80–97.

[27] C. Peikert and A. Rosen, "Efficient collision-resistant hashing from worst-case assumptions on cyclic lattices," in *TCC*. Springer, 2006, pp. 145–166.

[28] T. Pöppelmann, L. Ducas, and T. Güneysu, "Enhanced lattice-based signatures on reconfigurable hardware," in *CHES*, 2014, pp. 353–370, full version: https://eprint.iacr.org/2014/254.pdf.

[29] T. Pöppelmann and T. Güneysu, "Towards efficient arithmetic for lattice-based cryptography on reconfigurable hardware," in *LATINCRYPT*, 2012, pp. 139–158.

[30] T. Pöppelmann and T. Güneysu, "Towards practical lattice-based public-key encryption on reconfigurable hardware," in *SAC*, 2013, pp. 68–85.

[31] T. Pöppelmann and T. Güneysu, "Area optimization of lightweight lattice-based encryption on reconfigurable hardware," in *ISCAS*, 2014, pp. 2796–2799.

[32] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," in *STOC*, 2005, pp. 84–93.

[33] O. Reparaz, S. S. Roy, F. Vercauteren, and I. Verbauwhede, "A masked ring-LWE implementation," in *CHES*, 2015, pp. 683–702.

[34] 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.

[35] S. S. Roy, F. Vercauteren, and I. Verbauwhede, "High Precision Discrete Gaussian Sampling on FPGAs," in *SAC*, 2013, pp. 1–39.

[36] M.-J. O. Saarinen, "Gaussian sampling precision and information leakage in lattice cryptography," Cryptology ePrint Archive, Report 2015/953, 2015.

[37] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM J. Comput.*, vol. 26, no. 5, pp. 1484–1509, Oct. 1997.

[38] J. H. Silverman and W. Whyte, "Timing attacks on NTRUEncrypt via variation in the number of hash calls," in *CT-RSA*. Springer, 2007, pp. 208–224.

[39] J. Von Neumann, "Various techniques used in connection with random digits," *Applied Math Series*, vol. 12, no. 36-38, p. 1, 1951.

James Howe received first-class honours BSc degree in Mathematics at the University of Greenwich, and MSc degree in the Mathematics of Cryptography and Communications at Royal Holloway, University of London. He is a research assistant at the Centre for Secure Information Technologies (CSIT), Queen's University Belfast. His research interests include cryptographic hardware and software, public-key cryptography, post-quantum cryptography, and cryptanalysis.

Ayesha Khalid completed her B.E. in Computer Systems Engineering from National University of Sciences and Technology (NUST), Pakistan and her M.S. in Electrical Engineering from Center for Advanced Studies in Engineering (CASE), affiliated with University of Engineering and Technology, UET-Taxila, Pakistan. From 2008 to 2010, she served as a Lecturer in the Department of Electrical Engineering at Muhammad Ali Jinnah University, Islamabad and later joined RWTH Aachen, Germany for her doctoral studies. She is the recipient of DAAD scholarship award for PhD. Currently, she is a Research Fellow at Queens University Belfast.

Ciara Rafferty (M'14) received first-class honours in the BSc. degree in Mathematics with Extended Studies in Germany at Queen's University Belfast in 2011 and the Ph.D. degree in electrical and electronic engineering from Queen's University Belfast in 2015. She is currently a Research Assistant at Queen's University Belfast. Her research interests include hardware cryptographic designs for homomorphic encryption and lattice-based cryptography.

Francesco Regazzoni received the MS degree from Politecnico di Milano, Italy. He is a postdoctoral researcher at the ALaRI Institute of University of Lugano, Switzerland, where he completed the PhD degree. Previously, he has been an assistant researcher with the Crypto Group at Universit Catholique de Louvain (UCL) and at TU Delft. His research interests include embedded systems security, side channel attacks, cryptographic hardware, electronic design automation for security, and random number generators.

Máire O'Neill (M'03-SM'11) received the M.Eng. degree with distinction and the PhD degree in electrical and electronic engineering from Queen's University Belfast, U.K., in 1999 and 2002, respectively. She is a Chair of Information Security at Queen's and holds an EPSRC Leadership fellowship to conduct research into next generation data security architectures. She previously held a UK Royal Academy of Engineering research fellowship from 2003 to 2008. She has authored two research books and has more than 100 peer-reviewed conference and journal publications. Her research interests include hardware cryptographic architectures, side channel analysis, physical unclonable functions and post-quantum cryptography. In 2014 she was awarded a Royal Academy of Engineering Silver Medal, which recognises outstanding personal contribution by an early or mid-career engineer that has resulted in successful market exploitation.