I. INTRODUCTION

Cybersecurity is being shaped by the need to secure algorithms, data, devices, and networks under future threats as technology evolves [1, 2]. The present in-use public-key cryptosystems, such as RSA and elliptic curve cryptography (ECC), are vulnerable to the attack of the quantum computer using Shor’s algorithm [3]. Currently, the National Institute of Standards and Technology (NIST) is standardizing one or more quantum-resistant public-key cryptographic algorithms, which is known as post-quantum cryptography (PQC) [4]. These PQC schemes can be classified as key encapsulation mechanisms (KEM) based [5, 6] and digital signature based schemes [7, 8]. Several schemes relying on the lattice-based computational primitive are selected for the round 4 standardization.

Meanwhile, the hardware implementations and accelerations for the finalist candidates are required for evaluating the overall performance. In general, the accelerations of PQC schemes focus on modular polynomial multiplier, modular (integer) multiplier, hash module, and sampler [8, 9]. In particular, the efficient designs for modular polynomial multiplier have been extensively studied, using number theoretic transform (NTT) [10–12], schoolbook polynomial multiplication algorithm [13, 14], or Karatsuba algorithm [9, 15]. The architectures for modular multiplier and hash module have also been investigated in [16–19]. In contrast, hardware architectures for the sampler are less studied. The PQC schemes deploying the learning with errors (LWE) problem’s variants require adding the errors into the ciphertext to make the problem computationally hard. Such errors are typically generated from the sampler, which protects secure information.

Different from previous architectures that optimize the sampler architecture independently [20–25], this paper proposes a novel integral architecture by optimizing the sampler jointly with the modular polynomial multiplier, accompanied by a simple control unit. The sampler in the proposed design utilizes the abundant computation resources within the underlying modular polynomial multiplier to reduce the overall resource consumption. In particular, we design the Gaussian distribution samplers using the Knuth-Yao algorithm [26] and discrete Ziggurat algorithm [27], respectively, by following the proposed integral architecture.

The rest of this paper is organized as follows: Section II reviews the mathematical background for lattice-based cryptography and the corresponding hardware architectures in prior works. Section III presents the details of our hardware architecture design. The performance of our proposed architecture is provided and analyzed in Section IV. Finally, Section V presents remarks and concludes the paper.

II. BACKGROUND

LWE is based on the lattice problem, which is NP-hard for quantum computers [28]. Due to its arithmetic simplicity and strong security performance, PQC schemes based on the LWE problem’s successors are popular among the existing schemes. Crystals-Kyber (Module-LWE) [5] and NewHope (Ring-LWE) [6], are two representative LWE-based cryptosystems that have been considered for the NIST PQC standardization.

For a Ring-LWE based scheme, the polynomial computation is computed over ring $R_q = \mathbb{Z}_q/(x^n + 1)$, where $x^n + 1$ is an irreducible polynomial of degree (dimension) $n$, and the polynomial coefficient modulus is $q$. The Ring-LWE sample $(a(x), b(x)) \in R_q \times R_q$ is defined as follows: $a(x)$ is an uniformly random polynomial over ring $R_q$, and the corresponding $b(x)$ is expressed as

$$b(x) = a(x) \cdot s(x) + e(x) \in R_q,$$

where $s(x) \in R_q$ is the secret, and $e(x) \in R_q$ is the error term. Specifically, both polynomials are random polynomials sampled from a discrete Gaussian distribution with a standard deviation of $\sigma$ [29]. Notably, the construction of Ring-LWE and Module-LWE based variants are similar with the only...
difference of $s(x)$ and $a(x)$ are $d$-dimension vectors, i.e., $s(x), a(x) \in (\mathbb{F}_q)^d$, and the entries of $s(x)$ and $a(x)$ are polynomials in $\mathbb{F}_q$.

Meanwhile, the multiplication between two polynomials (vectors) over the ring is heavily used in LWE-based schemes. In the literature, the number-theoretic transform (NTT) based modular polynomial multiplication is widely used to reduce the quadratic complexity $O(n^2)$ from the schoolbook polynomial multiplication to $O(n \log n)$ [10]. The main procedure of the modular polynomial multiplication of two polynomials using NTT is to convert all the coefficients of the polynomials into the NTT domain. Then a direct coefficient-wise multiplication is performed between the two polynomials. Finally, the resulting polynomial uses an inverse NTT to recover the coefficients back to the original algebraic domain polynomial.

III. PROPOSED SAMPLER DESIGNS

In this section, we present the hardware architecture designs for two widely used sampling algorithms, i.e., the Knuth-Yao algorithm and the discrete Ziggurat algorithm.

A. High-level overview

Our design aims to reduce the hardware cost by sharing the arithmetic operations between the sampling and ring modular polynomial multiplication (i.e., NTT) to facilitate an integral architecture. Besides, since ring modular polynomial computations need the results from the sampler for key generation and encryption, sharing the hardware resources does not impact the overall computation flow. The difference between this work and prior works is shown in Fig. 1. Previous works typically separate sampler from the ring modular polynomial multiplier as depicted in Fig. 1(a). In contrast, we design the sampler as an integral part of the modular polynomial architecture as shown in Fig. 1(b). Here the sampler is redesigned as a sampling control module, where the sampling process relies on the results computed by the ring modular polynomial multiplier rather than computed within the sampler module. Hence, it can reduce the resource consumption of sampling algorithms involving multiplications and additions.

The sampling control module is responsible for providing the ring modular polynomial multiplication with the intermediate value during sampling computation and sampled value for key generation and encryption. The detailed reconfiguration process for a typical ring modular polynomial multiplication using NTT is shown in Fig. 2. The reconfiguration uses multiplexers (MUXes) to configure the NTT unit for ordinary NTT operation or reconfigured sampling operation. For sampling operation, the paths of the butterfly unit are only partially activated for addition or multiplication. The modular addition consists of a general addition and modular reduction, which is used to reduce the results to be smaller than $q$. Modular addition can be configured to general addition by bypassing modular reduction. General additions with longer bit-length than $q$ can be realized by cascading several general adders reconfigured from modular adders. For modular multiplications, it first multiplies two products and then reduces the final result to be less than $q$. Similarly, configuring modular multiplication to general multiplication can be realized by extracting the multiplication result without modular reductions. Given that the sampling algorithm could have large bit-length multiplications that cannot be accommodated within one modular polynomial multiplier unit (i.e., NTT unit), we need to partition them into small bit-length multiplications using general multipliers configured from modular multipliers. Then, we will add the partial results based on their actual locations within the large bit-length multiplication results to obtain the final product using general adders reconfigured from modular adders.

B. Knuth-Yao sampling control design

1) Sampling algorithm background: A tree-based sampling algorithm for non-uniform distributions was proposed in [26]. It builds a probability matrix $P$ for non-negative values’ probability for 0 to $N - 1$ as $p_0, \ldots, p_{N-1}$ represented with a maximum $\lambda$ bits individually. It can also be represented as a discrete distribution tree with maximum $\lambda$ levels with internal nodes that have two children and terminal nodes. The sampling process starts from the root of the tree to the terminal nodes by iteratively consuming a single random bit and stepping down one level until it reaches a terminal node. The integer label of the reached terminal node will be returned as the sampled value. The process is described in Algorithm 1.

Various hardware implementations and optimizations for the Knuth-Yao algorithm have been studied. For instance, a hardware-friendly random walk for discrete Gaussian sampler using the Kunth-Yao algorithm was implemented in [21]. A reconfigurable adder tree for the Hamming distance computation and loop unrolling for the sample search was proposed in [31]. A discrete Gaussian sampler exploits the $i$-th column/tree-level information to identify the nodes for tree traversal instead of visiting each level at the random walk was proposed in [32]. The observation based on the unique mapping between the input random bits and the output samples allows each output

---

**Fig. 1:** Comparison with prior works.
sample bit to be produced by a Boolean function of the input bit sequence, inducing the constant-time Gaussian sampling proposed in [22]. A custom combinational circuit generation for sampling from arbitrary complete discrete distributions was proposed in [23]. However, all these prior works design the sampler separately from the polynomial multiplier.

2) Modular polynomial multiplier configuration: The main computations of the Knuth-Yao algorithm are at line 3 and line 6 of Algorithm 1. Given $d = 2d + \lfloor r \cdot \text{cnt} \rfloor - \text{HD[col]}$, $\text{cnt} = \text{cnt} + 1$, and if $d < 0$ then $d = d + \text{P[row][col]}$, we can configure modular addition to accommodate the update procedure of $d$ by cascading several general adders. Additionally, as seen from the operation at line 3 in Algorithm 1, $d$ should always be less than $\lambda - 1$. By setting $\text{HD}_\text{sum} = \sum_{col=0}^{\lambda-1} \text{HD[col]}$ to have a valid sample value. It can further reduce the required total bit-length of general adders from $\lambda + 1$ bits to $(\log \text{HD}_\text{sum}) + 1$ bits. Even without the optimization, for example, for distribution with $r = 215$ and $q = 11289$ with the maximum sample value of $9r = 1935$, the maximum possible $\text{HD}_\text{sum}$ is $1935 \times 64 = 12840$, which can be represented in 14 bits.

3) Sampler control design: The overall hardware design for the Knuth-Yao algorithm is shown in Fig. 3. The control module that determines the sampling process consists of three components, i.e., $d < 0$ at line 4 by checking the most significant bit of $d$ that is represented in 2’s complement, $d = 0$ at line 7 in Algorithm 1, and $d > \text{HD}_\text{sum}$ to allow early stopping as discussed above. The value of $d$ is updated based on these three comparisons to determine the following update pattern, or output sampled value $(-1)^r \cdot \text{row}$ when $d = 0$.

C. Discrete Ziggurat sampling control design

1) Sampling algorithm background: Rejection sampling is another category of sampling algorithm applicable for an arbitrary probability distribution. Ziggurat sampling is an optimized form of rejection sampling by covering the probability density function with several horizontal rectangles with identical areas [33]. For the selected rectangle with index $i$, points to the left of $x_{i-1}$ are accepted, while each sampled point to the right need further computation based on its location over the probability distribution function to determine whether it gets accepted or not. The later computations generally incur a large overhead. The discrete Ziggurat [27] algorithm is presented in Algorithm 2 with sLine operation described in Algorithm 3.

One of the first hardware Ziggurat sampling implementations was proposed in [24]. The design pipelines the computation of rectangular regions in parallel with the evaluation of the wedge and tail, which are the areas underneath the probability distribution function to the right of $x_{i-1}$. A variant of Ziggurat sampling that uses trapezoids rather than rectangles to partition the distribution was proposed in [34]. The partition leaves fewer rejection areas with similar hardware performance compared with the rectangle partition. A parameterized design...
Algorithm 2 Discrete Ziggurat Algorithm [27]

Input: The number of rectangles $m$. Lower right corners for rectangles with coordinates $[x_i], y_i = [y_i]$ for $i \in m$ with precision $\lambda$.

Output: $s \cdot x$.

1: while True do
2:   choose rectangle $i \in \{1, \ldots, m\}$, sign $s \in \{-1, 1\}$, random bit $b \in \{0, 1\}$, choose random value $x \in \{0, \ldots, [x_i]\}$ and $y' \in \{0, \ldots, 2^\lambda - 1\}$
3:   if $0 < x \leq [x_i - 1]$ then
4:      return $s \cdot x$
5:   else
6:      if $x = 0$ and $b = 0$ then
7:         return $s \cdot x$
8:      else
9:         $y = y' \left(y_i - 1 - y_i\right)$
10:        $y_i = sLine([x_i - 1 - [x_i]] - y_i, x)$
11:        if $[x_i] + 1 \leq \sigma$ then
12:           if $\bar{y} \leq 2^\lambda \cdot y_i$ and $\bar{y} \leq 2^\lambda \cdot (p_x - y_i)$ then
13:              return $s \cdot x$
14:           else
15:              continue
16:           else
17:              if $\sigma \leq [x_i - 1]$ then
18:                 if $\bar{y} \geq 2^\lambda \cdot y_i$ and $\bar{y} \geq 2^\lambda \cdot (p_x - y_i)$ then
19:                    continue
20:                 else
21:                    return $s \cdot x$
22:                 else
23:                    if $\bar{y} \leq 2^\lambda (p_x - y_i)$ then
24:                       return $s \cdot x$
25:                    else
26:                       continue

Algorithm 3 sLine() Operation in Algorithm 2

Input: $[x_i - 1 - [x_i]], y_i - 1, y_i, x$

1: if $[x_i - 1] = [x_i]$ then
2:   return $-1$
3:   $y_i = \hat{y}_i, \hat{y}_i - 1 = \left[\begin{array}{c} 
\hat{y}_i - 1, i > 1 \\
1, i = 1 \end{array}\right]$
4: return $\frac{y_i - \hat{y}_i - 1}{[x_i] - [x_i - 1]} (x - [x_i])$

for Ziggurat sampling was proposed in [25] explores the trade-offs between the number of rectangles for partition and hardware resource consumption. Recently, a discrete Ziggurat sampling hardware implementation is proposed [30]. The detailed comparisons over acceptance and rejection regions are summarized in Algorithm 2. The sampling algorithm is the major computational bottleneck.

2) Modular polynomial multiplier configuration: For the discrete Ziggurat sampler, the major computation costs lie in sLine of Algorithm 3, which determines whether the sampled point is underneath the curve or not. Given that

Fig. 4: Discrete Ziggurat sampler hardware design. The shaded multiplier is computed within the ring modular polynomial module.

$$k = \frac{y_i - \hat{y}_i - 1}{[x_i] - [x_i - 1]}$$

is fixed for $x$ located within the selected partitioned rectangle, it reduces to $k(x - [x_i])$. Thus, it can be computed by reconfiguring the modular polynomial multiplier as discussed in Section III-A. The numbers of small bit-length general multiplications and general additions are determined by the size of $q$ and the probability distribution that the sampler follows. The reconfiguration is trivial. For $\sigma = 3.33, q = 4093, n = 512$, the maximum possible sampled value for the sampler is $30 \approx 9\sigma$. For a 64-bit $k$, 64-bit $x$-bit multiplication is needed for the sLine operation. Given that 5-bit is smaller than $\log_2(4093) = 12$ bit, we partition the 64 bits into 6 12-bit partial multiplications with an output bit-length of 17. Then we need to accommodate 5 general additions of bit-length 12 to obtain the final result. Each result of the 17-bit multiplications is partitioned into higher 5 bits and lower 12 bits, while the higher 5 bits are added to the next lower 12 bits. When $n = 512$, we have 9 NTT units available for the modular polynomial multiplication. Besides, we have two modular multipliers before the NTT units as part of the negative wrapped convolution as discussed in [10]. To accommodate the multiplication, we reconfigure the two modular multipliers for the negative wrapped convolution as the general multiplier and then reconfigure the first 4 NTT units as general multipliers and the last 5 NTT units as general adders. Other computations in Algorithm 2 use pre-computed values for further reducing the resource consumption, similar to the approach in [20].

3) Sampler control design: The overall hardware design for the discrete Ziggurat algorithm is shown in Fig. 4. Similarly, the discrete Ziggurat sampler control module involves multiple comparisons to determine whether the sampled value is accepted or not. Each acceptable condition is inferred from the comparison results based on the computations described in Algorithm 2. For the comparisons at line 11 and line 17, the results are determined by the selected rectangle index $i$. A stored bit value for an individual rectangle index $i$ can be
used to avoid actual comparisons. Besides, the determinations of acceptance at lines 3, 6, 12, 18, and 23 in Algorithm 2 are done by the comparisons. All Boolean comparison results are logically OR-ed to obtain the final accepted sample value.

IV. EXPERIMENTAL RESULTS

In our experiment, we adapt the design in [10] as our modular polynomial multiplier module. The sampler controller is fed with uniformly sampled random bits. We implement the Knuth-Yao algorithm and the discrete Ziggurat algorithm using two sets of LWE parameters \( \sigma_{LP} = 3.33 \) w/o. config \( \sigma_{BLISS} = 215 \) w/o. config \( \sigma_{LP} = 3.33 \) w. config \( \sigma_{BLISS} = 215 \) w. config

| Distribution parameter and setting | Module | LUT/FF/Slice | BRAM/DSP |
|-----------------------------------|--------|-------------|----------|
| \( \sigma_{LP} = 3.33 \) w/o. config | NTT    | 13/0/1/120/1476 | 0/18     |
| Sampler                           |        | 64/64/28     |          |
| \( \sigma_{BLISS} = 215 \) w/o. config | NTT    | 3486/2109/3382 | 0/10     |
| Sampler                           |        | 94/41/94     |          |
| \( \sigma_{LP} = 3.33 \) w. config Reconfigured NTT | NTT    | 1323/1628/1491 | 0/18     |
| Sampler Control                   |        | 51/43/71     | 2/0      |
| \( \sigma_{BLISS} = 215 \) w. config Reconfigured NTT | NTT    | 3513/2371/3865 | 0/10     |
| Sampler Control                   |        | 43/51/29     | 7/0      |

| Distribution parameter and setting | Module | LUT/FF/Slice | BRAM/DSP |
|-----------------------------------|--------|-------------|----------|
| \( \sigma_{LP} = 3.33 \) w/o. config | NTT    | 1303/172/1475 | 0/18     |
| Sampler                           |        | 341/341/100  | 0/17     |
| \( \sigma_{BLISS} = 215 \) w/o. config | NTT    | 3625/2218/3991 | 0/10     |
| Sampler                           |        | 281/107/281  | 6/16     |
| \( \sigma_{LP} = 3.33 \) w. config Reconfigured NTT | NTT    | 1511/1616/1679 | 0/18     |
| Sampler Control                   |        | 211/10/211   | 0/1      |
| \( \sigma_{BLISS} = 215 \) w. config Reconfigured NTT | NTT    | 3756/2258/4122 | 0/10     |
| Sampler Control                   |        | 181/19/123   | 6/2      |

V. CONCLUSION

This paper presented an integral hardware architecture for lattice-based cryptography that integrates the sampler with the underlying ring polynomial computation unit. This design implements the widely used Knuth-Yao algorithm and Ziggurat algorithms for discrete Gaussian sampling.

REFERENCES

[1] Y. Lao and K. K. Parhi, “Protecting DSP circuits through obfuscation,” in 2014 IEEE International Symposium on Circuits and Systems (ISCAS), 2014, pp. 798–801.
[2] J. Clements and Y. Lao, “Backdoor attacks on neural network operations,” in 2018 IEEE Global Conference on Signal and Information Processing (GlobalSIP), 2018, pp. 1154–1158.
[3] P. W. Shor, “Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer,” SIAM review, vol. 41, no. 2, pp. 303–332, 1999.
[4] D. J. Bernstein and T. Lange, “Post-quantum cryptography,” Nature, vol. 549, no. 7671, pp. 188–194, 2017.
[5] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé, “CRYSTALS-Kyber: a CCA-secure module-lattice-based KEM,” in 2018 IEEE European Symposium on Security and Privacy (EuroS&P). IEEE, 2018, pp. 353–367.
[6] E. Alkim, L. Ducas, T. Poppelmann, and P. Schwabe, “Post-quantum key {Exchange—A} new hope,” in 25th USENIX Security Symposium (USENIX Security 25), 2016, pp. 327–343.
[7] L. Ducas, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé, “Crystals–dilithium: Digital signatures from module lattices,” 2018.
[8] U. Banerjee, T. S. Ukyab, and A. P. Chandrakasan, “Sapphire: A configurable crypto-processor for post-quantum lattice-based protocols,” IACR Transactions on Cryptographic Hardware and Embedded Systems, pp. 17–61, 2019.
[9] Y. Zhu, M. Zhu, B. Yang, W. Zhu, C. Deng, C. Chen, S. Wei, and L. Liu, “LWRpro: An energy-efficient configurable crypto-processor for module-lwr,” IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 68, no. 3, pp. 1146–1159, 2021.
[10] W. Tan, A. Wang, Y. Lao, X. Zhang, and K. K. Parhi,
“Pipelined high-throughput NTT architecture for lattice-based cryptography,” in 2021 Asian Hardware Oriented Security and Trust Symposium (AsianHOST). IEEE, 2021, pp. 1–4.

[11] T. Oder and T. G"uneysu, “Implementing the NewHope-Simple key exchange on low-cost FPGAs,” in International Conference on Cryptology and Information Security in Latin America. Springer, 2017, pp. 128–142.

[12] P.-C. Kuo, W.-D. Li, Y.-W. Chen, Y.-C. Hsu, B.-Y. Peng, C.-M. Cheng, and B.-Y. Yang, “High performance post-quantum key exchange on FPGAs,” Journal of Information Science and Engineering, vol. 37, no. 5, pp. 1211–1229, 2021.

[13] S. S. Roy and A. Basso, “High-speed instruction-set co-processor for lattice-based key encapsulation mechanism: Saber in hardware,” IACR Transactions on Cryptographic Hardware and Embedded Systems, pp. 443–466, 2020.

[14] J. Xie, P. He, and C.-Y. Lee, “CROP: FPGA implementation of high-performance polynomial multiplication in saber kem based on novel cyclic-row oriented processing strategy,” in 2021 IEEE 39th International Conference on Computer Design (ICCD). IEEE, 2021, pp. 130–137.

[15] J. M. B. Mera, F. Turan, A. Karmakar, S. S. Roy, and I. Verbraeudhe, “Compact domain-specific co-processor for accelerating module lattice-based KEM,” in 2020 57th ACM/IEEE Design Automation Conference (DAC). IEEE, 2020, pp. 1–6.

[16] R. Paludo and L. Sousa, “NTT architecture for a linux-ready RISC-V fully-homomorphic encryption accelerator,” IEEE Transactions on Circuits and Systems I: Regular Papers, pp. 1–14, 2022.

[17] W. Tan, B. M. Case, A. Wang, S. Gao, and Y. Lao, “High-speed modular multiplier for lattice-based cryptosystems,” IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 68, no. 8, pp. 2927–2931, 2021.

[18] M. Sundal and R. Chaves, “Efficient FPGA implementation of the SHA-3 hash function,” in International Symposium on VLSI (ISVLSI). IEEE, 2017, pp. 86–91.

[19] W. Tan, A. Wang, Y. Lao, X. Zhang, and K. K. Parhi, “Low-latency VLSI architectures for modular polynomial multiplication via fast filtering and applications to lattice-based cryptography,” arXiv preprint 2110.12127, 2021.

[20] J. Howe, T. Oder, M. Krausz, and T. G"uneysu, “Standard lattice-based key encapsulation on embedded devices,” IACR Transactions on Cryptographic Hardware and Embedded Systems, pp. 372–393, 2018.

[21] S. Sinha Roy, F. Vercauteren, and I. Verbaeudhe, “High precision discrete gaussian sampling on FPGAs,” in International Conference on Selected Areas in Cryptography. Springer, 2013, pp. 383–401.

[22] A. Karmakar, S. S. Roy, O. Reparaz, F. Vercauteren, and I. Verbaeudhe, “Constant-time discrete gaussian sampling,” IEEE Transactions on Computers, vol. 67, no. 11, pp. 1561–1571, 2018.

[23] M. X. Lyons and K. Gaj, “Sampling from discrete distributions in combinational hardware with application to post-quantum cryptography,” in 2020 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, 2020, pp. 610–613.

[24] G. Zhang, P. H. W. Leong, D.-U. Lee, J. D. Villasenor, R. C. Cheung, and W. Luk, “Ziggurat-based hardware gaussian random number generator,” in International Conference on Field Programmable Logic and Applications, 2005. IEEE, 2005, pp. 275–280.

[25] R. Agrawal, L. Bu, and M. A. Kinsy, “A post-quantum secure discrete gaussian noise sampler,” in 2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST). IEEE, 2020, pp. 295–304.

[26] D. E. Knuth and A. C.-C. Yao, “The complexity of nonuniform random number generation,” Algorithm and Complexity, New Directions and Results, pp. 357–428, 1976.

[27] 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 International Conference on Selected Areas in Cryptography. Springer, 2013, pp. 402–417.

[28] O. Regev, “On lattices, learning with errors, random linear codes, and cryptography,” Journal of the ACM (JACM), vol. 56, no. 6, pp. 1–40, 2009.

[29] V. Lyubashevsky, C. Peikert, and O. Regev, “On ideal lattices and learning with errors over rings,” in Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer, 2010, pp. 1–23.

[30] J. Howe, A. Khalid, C. Rafferty, F. Regazzoni, and M. O’Neill, “On practical discrete gaussian samplers for lattice-based cryptography,” IEEE Transactions on Computers, vol. 67, no. 3, pp. 322–334, 2016.

[31] S. Zhao, N. Shah, W. Meert, and M. Verhelst, “Discrete samplers for approximate inference in probabilistic machine learning,” in 2022 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, 2022, pp. 1221–1226.

[32] S. S. Roy, F. Vercauteren, N. Mentens, D. D. Chen, and I. Verbaeudhe, “Compact ring-LWE cryptoprocessor,” in International workshop on cryptographic hardware and embedded systems. Springer, 2014, pp. 371–391.

[33] G. Marsaglia and W. W. Tsang, “The ziggurat method for generating random variables,” Journal of statistical software, vol. 5, pp. 1–7, 2000.

[34] H. Edrees, B. Cheung, M. Sandora, D. B. Nummey, and D. Stefan, “Hardware-optimized ziggurat algorithm for high-speed gaussian random number generators,” in ERS, 2009, pp. 254–260.

[35] R. Lindner and C. Peikert, “Better key sizes (and attacks) for LWE-based encryption,” in Cryptographers’ Track at the RSA Conference. Springer, 2011, pp. 319–339.

[36] L. Ducas, A. Durmus, T. Lepoint, and V. Lyubashevsky, “Lattice signatures and bimodal gaussians,” in Annual Cryptology Conference. Springer, 2013, pp. 40–56.