Compact and Secure Generic Discrete Gaussian Sampler based on HW/SW Co-design

Sudarshan Sharma*, Arnab Bag†‡, Debdeep Mukhopadhyay†§
* Dept. of Electronics and Electrical Communication Engineering,† Dept. of Computer Science and Engineering
IIT Kharagpur, Kharagpur, India
Email:{*sudarshansharma04, ‡arnabbag}@iitkgp.ac.in, §debdeep@cse.iitkgp.ac.in

***Abstract*—In this paper, we present the first Hardware (HW) / Software (SW) co-design based generic discrete Gaussian sampler architecture on the Xilinx Zynq platform. The area optimized and secure sampler can produce a distribution based on an arbitrary standard deviation and center given as input. We use multi-level logic optimization on Knuth-Yao algorithm's Discrete Distribution Generating (DDG) tree travel-based Boolean mapping of random bits and samples instead of the previous two-level logic optimization to reduce the resource utilization. This method results in nearly 60% lesser LUT utilization compared to the previous designs on Xilinx FPGAs. Further, we introduce improvements in the shuffling algorithm leveraging the HW/SW co-design methodology compared to the existing shuffling architectures for randomizing Gaussian samples to protect against timing-based side-channel attacks.**

***Index Terms*—Discrete Gaussian Sampler, HW/SW Co-design, Knuth-Yao Algorithm, Shuffling based countermeasure, multi-level logic optimization**

I. Introduction

Recent improvements in the field of quantum computers [1] pose a severe threat to existing standard Public Key Cryptosystems (PKC), including Elliptic Curve Cryptography (ECC) and Rivest – Shamir – Adleman (RSA) based on the Elliptic Curve Discrete Logarithm Problem (ECDLP) and the Integer Factoring Problem (IFP) respectively. Large scale quantum computers using the Shor's [2] and Proos-Zalka's [3] algorithms can break the inherent hard problems associated with these schemes. Lattice-based cryptosystem has become a promising replacement in the post-quantum computing era which are secure against the above attacks. Presently, there are no known quantum algorithms for solving the hard lattice problems, thus opening up the era for post-quantum cryptographic constructions. Some of the hard problems involving lattices are the shortest vector problem (SVP), its approximate counterpart approximate-SVP, and bounded decision decoding (BDD) [4]. The Learning with errors (LWE) problem [5] and short integer solution (SIS) problem [6] deduce to approximate-SVP and SVP, respectively. Note that these hard problems involving lattices cannot be solved in polynomial time using any classical algorithm. Moreover, the cryptographic constructions through these hard lattice problems have worst-case hardness compared to the classical constructions resulting in a stronger security guarantee.

Discrete Gaussian sampler is considered as the heart of Lattice-based cryptography. It plays a vital role in the worst-case security guarantee of Lattice-based constructions. Efficient (either area, speed or both) discrete Gaussian sampler is necessary to implement a Lattice-based cryptosystem in practice. NIST's recent efforts for quantum-resistant public-key cryptographic algorithms standardization further strengthens this requirement. However, developing a robust and generic discrete Gaussian sampler is a non-trivial task which involves high-precision architecture design, optimising pre-computed table access and execution, and finally, mitigating side-channel vulnerabilities. We aim to provide a design that covers these design aspects without compromising on the security.

A number of designs [7], [8], [9], [10], [11] have focused on the various aspects of discrete Gaussian sampler design. However, most of these implementations are application-specific and have fixed parameters (no modularity and scalability). Also, the resource usage for side-channel mitigation is prohibitively high. Therefore, such shortcomings in the existing architectures expedite the need for robust and modular sampler implementation. Also, at the same time, offers suitable performance with improved security for modern applications.

As a major contribution of this work we propose, to the best of our knowledge, the first HW/SW co-design based Gaussian sampler that can generate samples for an arbitrary center. We provide a novel multi-level logic minimisation strategy used to develop the compact processing logic. Moreover, we incorporate an improved shuffling process to mitigate side channel attacks.

The remainder of the paper is organized as follows– Section II provides a brief insight into the required mathematical relations. The overview and the associated algorithms for generic discrete Gaussian sampler are described in Section III. Section IV covers the implementation methodology of the sub-blocks, along with the combined architecture. Experimental Setup and Results are explained in Section V.

II. Preliminaries

The discrete Gaussian distribution on $\mathbb{Z}$ with a standard deviation of $\sigma > 0$ and mean c is defined as in Equation (1), where E is a random variable in $\mathbb{Z}$. The normalisation factor S approximated as $\sigma\sqrt{2\pi}$. The short-hand representation as $\mathbb{D}_{c,\sigma}$ for Equation 1 is used throughout the paper.

$$Pr(E = z) = \frac{e^{\frac{-(x-c)^2}{2\sigma^2}}}{S} \quad (1) \qquad \eta_\epsilon(\mathbb{Z}) \leq \sqrt{\frac{ln(2+\frac{2}{\epsilon})}{\pi}} \quad (2)$$

978-1-7281-8952-9/20/$31.00

The soothing parameter $\eta_\epsilon(\mathbb{Z})$ for an $\epsilon \in \mathbb{R}^+$ is defined over a lattice as the smallest $\sigma\sqrt{2\pi} > 0$ such that $S \times \mathbb{D}_{0,\sigma\sqrt{2\pi}} \leq 1 + \epsilon$.

Typically $\eta_\epsilon(\mathbb{Z}) < 6$ for very small $\epsilon < 2^{-160}$ [7]. A tail-cut factor τ is used to fix the number of samples, the typical interval of the sample is $[-\tau\sigma, \tau\sigma]$ ignoring the other values. This is also termed as the tail bound in some cases. Another important metric involved in the discrete Gaussian distribution is the precision of the probabilities that directly influence the accuracy and the security of the samplers.

III. Generic Gaussian Sampler Construction

LWE-based cryptography has varied requirement for Gaussian sampling algorithms. A zero-mean with a non-zero standard deviation is a popular choice with a wide range of spread, whereas non-zero mean distributions are necessary in applications like lattice trapdoor sampling [12]. In a such diversified requirement, an intuitive idea is to combine the samples from a smaller distribution and generate samples with higher standard deviation. A series of work has been done in these lines [13], [14], [7]. The authors in [14] proposed a recursive algorithm that combines samples from $\sqrt{s}$ using scalar multiples, where s is the required standard deviation. The Kullback-Liebler divergence (which is Rényi divergence of order one) was used instead of the statistical distance to prove the security estimates and closeness of the approximate sampler. Karmakar et al. [10] used this with two levels of recursion to generate Gaussian distribution with a large standard deviation. Further, Micciancio et al. [7] extended this work with a series of improvements. The algorithm proposed by Micciancio et al. [7] can be used to sample any discrete Gaussian distribution with an arbitrary center and standard distribution using multiple fixed smaller distributions. These fixed distributions were variable in the previous methods depending on the required standard deviation and center. Secondly, a new metric was introduced to get better security estimates, related to the standard notion of relative error and the Rényi divergence of order ∞. Finally, the authors suggest an online and offline phased constant-time implementation with the fixed sampler running in constant time offline phase and the recombination of obtained samples in run time, which also executes in constant time. However, they do not propose any design methodologies, feasibility metrics, and strategies to incorporate the fixed samplers architecture in hardware along with the details on how these online/offline phases interact. In the following section, we briefly revisit the algorithm proposed by Micciancio et al. [7].

A. Overview of Generic Gaussian Sampler

First, the samples are generated from the Base Samplers, which has a fixed standard deviation (reasonably small) and center. The generated samples are then combined using a convolution algorithm, described in the later sections. Finally, to achieve any arbitrary center, the center's binary representation is rounded using the samples from the Base Sampler. The rounding operation involves geometrical scaling of the center at every iteration until the integer sample is obtained. The Gaussian Sampler can be divided into two main parts – the Base Sampler (fixed) and the Rounder, which combine the samples from the fixed sampler using the convolution algorithm, and then uses Gaussian rounding to obtain the sample with given standard distribution and center. In the following sections, we will first discuss the Rounder and subsequently to the Base Sampler.

B. Rounder

Algorithm 1 $SampleI(i)$ [7]

1: **if** $i = 0$ **then**
2: $x \leftarrow SampleB_{\sigma_0}(0)$;
3: return x;
4: **else**
5: $x_1 \leftarrow SampleI(i-1)$;
6: $x_2 \leftarrow SampleI(i-1)$;
7: $y \leftarrow z_i x_1 + max(1, z_i - 1)x_2$;
8: return y;
9: **end if**
10: *Precomputed Values*
11: ${\sigma_i}^2 \leftarrow ({z_i}^2, max((z_i - 1)^2, 1)){\sigma_{i-1}}^2$
12: $zi \leftarrow \lfloor \frac{\sigma_{i-1}}{\sqrt{2}\eta_\epsilon(\mathbb{Z})} \rfloor$

The iterative combination ($SampleI(i)$ in Algorithm 1) of the samples with convolution results in a sample belonging to a larger standard deviation. The constraint $\sigma_o \geq \sqrt{2}\eta_\epsilon(\mathbb{Z})$ dictates the Base Sampler design parameters denoted as $SampleB_{\sigma_0}(0)$ for standard deviation σ_0 and center 0. Further, the convolution step is depicted in line 7 in Algorithm 1. The algorithm is recursive in nature, which makes the hardware implementation challenging.

The obtained samples are used by the main routine responsible of producing samples $SampleZ_{b,k,max}(\sigma, c)$ Algorithm 2 (where b is the base, k is the binary precision upon which center rounding starts, max is the maximum value of i corresponding to the number of standard deviation levels) which in turn uses the Gaussian center rounding routine $SampleC_b$, $(c \in b_k\mathbb{Z})$. In this work, we consider the value of the base b as 2, which means two-base samplers with centers 0 and 0.5. This results in minimum resource utilization in the hardware; however, it increases the required iterations to produce a sample in the software.

Algorithm 2 $SampleZ_{b,k,max}(\sigma, \text{c})$ [7]

1: $x \leftarrow SampleI(max)$
2: $K \leftarrow \sqrt{s^2 - \bar{s}^2}/s_{max}$
3: $\tilde{c} \leftarrow \lfloor c + Kx \rceil_k$
4: $y \leftarrow SampleC_b(\tilde{c})$
5: return y

Here, $\bar{s} = s_0(\sum_{i=0}^{k-1} b^{-2i})$ with s_0 being the fixed Base Sampler standard deviation. Note that if the value of k is too high, a larger number of samples are required for the Gaussian rounding of the center, and hence the execution time of the sampler would increase. Thus, this execution time is reduced using the optimization, which involves rounding the

least significant bits through a simple biased coin flip. Then the remaining k bits are rounded through the Gaussian rounding described below. The biased coin flip rounding is denoted by $\lfloor . \rceil_k$.

Algorithm 3 $SampleC_b,\ (c \in b_k\mathbb{Z})$ [7]

1: **if** $k = 0$ **then**
2: return 0
3: **else**
4: $g \leftarrow b^{-k+1}.SampleB_{\sigma_0}(b^{k-1}c)$;
5: return $g + SampleC_b(c - g \in b^{-k+1}\mathbb{Z})$;
6: **end if**

To understand Algorithm 3 consider a toy example as shown below.
$c = 0.0011011b_k \in 2^{-k}\mathbb{Z}$ (b_k is the k'th bit from the decimal)
$x = SampleB_{\sigma_0}(b_k)$
$x = 2x$ (LSB of x becomes 0)
$x = (x + b_k)/2^k$ (b_k at the LSB of x, c and x are aligned)
$c = c - x$
$c = 0.0011001 \in 2^{-k+1}\mathbb{Z}$
Here b is taken as 2.

C. Base Sampler

The available methods are rejection sampling [15], discrete Ziggurat sampling [16], Bernoulli sampling [17], cumulative distribution table (CDT) based sampling (inversion sampling) [13], Karney's algorithm [18] and Knuth Yao sampling [19]. Among the mentioned methods, CDT and Knuth Yao sampling are the popular ones [20] considering the security, performance, and storage metrics. Besides, Knuth Yao sampling surpasses the available alternatives in terms of efficiency, low entropy consumption, and time-independent implementation attributes (power, performance, and area trade-off).

1) Knuth Yao sampling: D. Knuth and A. Yao [19] proposed the Knuth Yao algorithm in 1976. The algorithm essentially traverses the Discrete Distribution Generating (DDG) tree depending on the random bits and returns a sample when it hits the terminal node of the tree first time. DDG is created based on a probability matrix consisting of k-bit floating points probability values arranged row-wise, depending on the number of samples. As seen in Fig. 1 the construction process is based on the property-at any level i of the tree from the root number of the terminal node is the Hamming weight of the ith column in the probability matrix. The traversal on the DDG tree starts from the root, and depending on the random bit (0/1), the left or right node is selected. The sampling process concludes when it hits the terminal node, the row index corresponding to the terminal node is the output. Intuitively, the probabilities with a higher magnitude will be closer to the root (one in the initial columns).

This algorithm was first used for discrete Gaussian sampling by Dwarakanath et al. [21]. A series of work followed with Roy et al. [8] proposing the first hardware architecture based on column scanning a lookalike to the exact definition. In a subsequent work, the author tried to mitigate the algorithm's data-dependent execution time using a two-step sampling approach with a shuffling technique. Karmakar et al. [10] presented the Boolean mapping between the random bit string and the sample as an alternative to the column scanning approach. The evaluation of the Boolean expression obtained from the mapping in real-time eliminated the data-dependent nature and paved the way for constant-time implementation. Additionally, in the subsequent work [11], they observed the unique property between the random bit string and the sample to reduce the program memory. According to the property the all the input random bit string which maps to the sample are of the form According to the property the whole random bit string of the input which maps to the sample are of the form $x^i(0/1)^i01^k$ where $i + j + k = n$ (x stands for don't care, 0/1 means 0 or 1, b^i means the string of b's of length i and n is the input random bit string). The efficient optimization of the Boolean expression is done by creating lists based on this property. The authors do not delve deep into the hardware level implementation, optimization involved, and present a comparative analysis with the available software alternatives. Furthermore, the design also requires a significant program memory for broad standard distribution in the software setting.

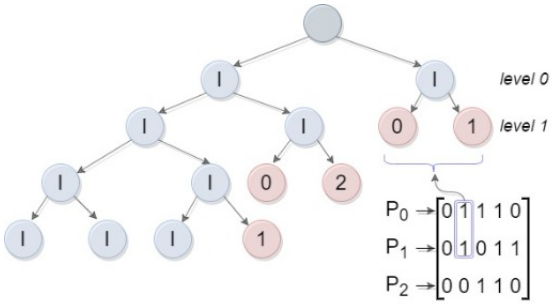

Fig. 1. DDG Tree using Knuth Yao Algorithm

IV. Implementation Methodology

As stated in the previous sections, the generic Gaussian Sampler can be efficiently implemented in Hardware (HW)/Software (SW) co-design fashion. HW/SW co-design approach brings the best of the two worlds - the flexibility of SW required to implement the recursive Rounder and the power/performance/area efficiency of HW for the Base Sampler. We used Digilent Zedboard for our experiment, which has a Zynq-7000 series FPGA as Programmable Logic (PL) coupled with Dual-core ARM Cortex-A9 as Processing Subsystem(PS).

We choose the Knuth Yao algorithm for the Base sampler implementation. Further, we stick to the state-of-the-art Boolean mapping method proposed in [10]. However, we introduce a multi-level logic optimization scheme along with a lightweight countermeasure leveraging the HW/SW co-design architecture to mitigate the side-channel vulnerability and reduce resource utilization.

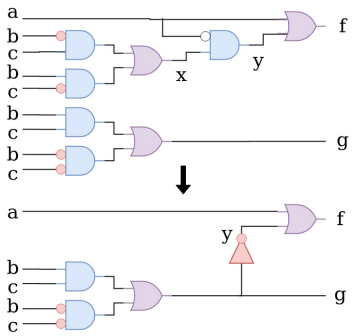

Fig. 2. Multilevel logic optimisation example[23].

The Boolean mapping obtained through the DDG tree traversal is a many-to-one mapping, i.e., many input random string points to one output sample. The authors in [10] used a two-level logic optimization technique to obtain a constant-time implementation of the Boolean mapping using ESPRESSO [22]. We use multi-level logic optimization on the entire Boolean mapping obtained through the DDG traversal in this work to minimize the overall area. Previously, the Boolean expression for each bit of the sample output was used to implement the sampler—these expressions where further multiplexed, resulting in significant overhead. However, using this method, one can generate a single Boolean network to generate all output bits. Furthermore, final implementation with the two-level optimization for all the sample output bits will have redundant logic blocks. The network obtained re-utilizes those intermediate redundant logic and results in nearly 60% lower LUT utilisation during implementation.

The two-level optimization used previously minimizes the number of Sum of Products (SOP) terms. In contrast, multi-level logic optimization minimizes the number of literals in the logic expression in a multi-level circuit, thus lowering the resource consumption. Moreover, the multi-level logic circuits are represented as Boolean networks where each node represents a gate or cell. These gate/cells are optimized by exploiting the don't care conditions existing in a particular region in the network chosen for optimization based on heuristics. The main advantage of using multi-level logic optimization is the re-utilization of logic created previously. For example, [23], considering the following logic function, which cannot be further optimized by ESPRESSO [22] (two-level logic optimization), is shown in the first half of Fig. 2, $f = a + y, g = b.c + \bar{b}.\bar{c}, y = \bar{a}.x, x = \bar{b}.c + b.\bar{c}$. Upon extracting the hidden don't care at logic gates, the multi-level logic optimization yields an optimized circuit. For the second half of the Fig. 2 this technique will result in lesser resource utilization of the entire Boolean mapping.

It is also assumed that multi-level logic synthesis works well for control-dominated circuits compared to the arithmetic circuits. ABC [24] is a state-of-the-art open-source multi-level logic optimization tool which incorporates And Inverted Graph (AIG) data structure for logic synthesis and verification. Besides, ABC is a complete logic synthesis package that facilitates LUT mapping, making the implementation flow less cumbersome. Note that the final logic circuit implemented using this methodology will not be constant time. We propose a lightweight countermeasure and discuss the side-channel countermeasure in later sections.

1) Base Sampler Implementation Technique: Since the Gaussian sampler is symmetric around the center, only one-half of the samples need to be considered. The other half can be generated using an additional random bit. First, the truth table of the Boolean mapping between the random bit string and the output sample is acquired according to the tehchniques discussed in [10]. Further, the multi-level logic optimization and Look-Up Table (LUT) mappings are done using the ABC tool. Two commands used are *strash* to convert the truth table to AIG and *if* to map the AIG to LUTs. Finally, the generated Verilog file, along with the TRNG, is prototyped on an FPGA. It is important to note that the Knuth Yao algorithm converges if the sum of all the sample's probability sums up to 1 (P_{sum}=1). The above criteria can be incorporated by adding a row in the probability matrix with a value of $1 - P_{sum}$, this sample is not considered in as a sample and is rejected if obtained. Moreover, this rejection of the sample occurs with very low probability since we have a larger tail bound. For $\sigma = 6.15543$ with tail bound as 9 and precision as 128, this probability is 183.8×10^{-27}.

2) Lightweight Countermeasure: The samples from the Base Sampler are time-dependent; thus, it is vulnerable to side-channel attacks. Roy et al. [9] proposed a lightweight one stage shuffling-based countermeasure using Fisher-Yates [25]/Knuth shuffle [19]. The technique was adopted by Saarinen [26], where he performed shuffling multiple times along with Gaussian convolution to obtain distribution with higher noise. Pessl [27] proposed an attack on both the architectures and concluded that with an increase in the stages of shuffling, the attack's complexity increase. We present three sets of improvement in shuffling based countermeasures.

- First, we incorporate a parallel shuffling network using the permutation network generator proposed in [28].
- Second, we remove the shuffling from the critical path of the sample generation utilizing the HW/SW co-design model, thus implementing it on the FPGA.
- Finally, we increase the shuffling stages by performing shuffling every time the sample is utilized by the Rounder, be it for the convolution or the center rounding.

Since the SW implementation is constant time and independent of the shuffling, this increases the complexity of attack. The permutation network takes one cycle and has a depth of O($\log_2 n$) and a size of O($n \log_2 n$) (similar to the cost of barrel shifter) where n is the number of samples to be shuffled. When compared to the popular Fisher-Yate/Knuth shuffling algorithm, which has the complexity of O(n) and executes serially, the permutation network has a complexity of O($\log_2 n$) and can be implemented in a parallel fashion. As seen in Fig. 3.(a) the permutation network consists of swap module as building blocks that swap the input based on the random bit. Fig. 3.(b) shows the shuffler architecture

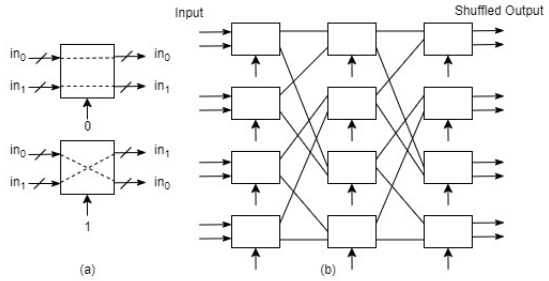

Fig. 3. (a) Permutation network building block (swapper) (b) Permutation network for 8 inputs.

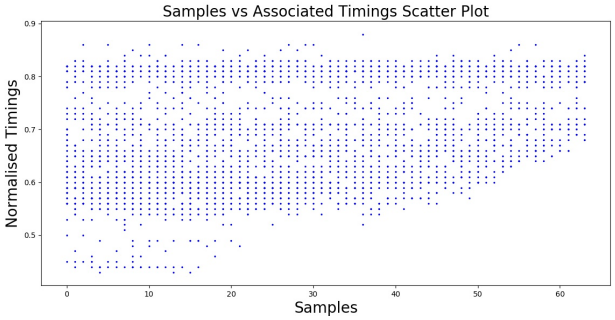

Fig. 4. Samples vs Timings Scatter Plot

for eight inputs, this can be extended depending on the number of samples in a batch.

Since the Boolean mapping is many-to-one, each sample has multiple timing leakages related to different inputs, as seen in Fig. 4 which shows the normalized timing for every path corresponding to a particular sample. The permutation network obfuscates the timing leakages by randomizing the samples, thus increasing the attack complexity. As an alternative, masked schemes based on binomial samplers are being explored [29] to secure lattice based cryptosystems.

3) Combined Architecture: The Rounder on the Zynq PS is time-independent and requires the same number of base samples and convolution for any arbitrary center and standard deviation. We generate a batch of 32 samples from the Base Samplers (both c_0 and c_1) in the PL, and then perform shuffling every time these samples are accessed. The shuffling algorithm returns a batch of shuffled indices from [0, 31], which is further used by the Rounder to access one of the base sample depending on the iteration index. We use multiple ES-TRNG [30] in PL to generate random numbers for Base Samplers, Shuffler in HW and Rounder in SW. All the modules in the PL are connected to the PS using AXI4-Lite interface, as depicted in Fig. 5.

V. Experiment Setup and Results

The parameters used for the Rounder implemented in the PS are as follows: $b = 2$, $max\_sigma\_level = 4$ *(defines the maximum sigma obtained from recursive convolution)* $\eta_\epsilon(\mathbb{Z}) = 2.5$, $\tau = 9$ *(tail-cut)*, $\lambda = 128$ *(binary representation precision)*, $maxflips = 35$ *(biased coin flip)*, $\sigma_0 = 6.1553$ *(Knuth Yao Sampler)*. The Knuth Yao Base Sampler for two centers $(0, 0.5)$ are implemented with the same parameters as the Rounder. The samples form both the Base Samplers are obtained in batches of 32. These batches are shuffled every time one of the samples is used. The shuffling algorithm implemented in the PL returns the shuffled indexes, and then the data corresponding to the shuffled index is used in that particular iteration. We use Digilent Zedboard as the main platform for evaluating the combined architecture. Moreover, Xilinx Vivado 2018.3 are used as the primary tool for implementation and testing.

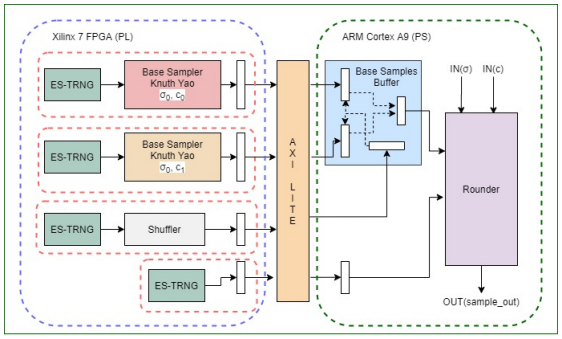

Fig. 5. Combined Architecture using Zynq based HW/SW codesign.

Table I shows the Base Sampler comparison with the existing state of the art architectures. For $\sigma = 3.33$ and $\lambda = 64$, our architecture does not require any memory elements (BRAM and FF), resulting in lower resource consumption. Finally, for second case with $\sigma = 6.15543$ and $\lambda = 112$ the implementation results in nearly 60% lesser resource utilisation. Table II presents the area and performance of various design modules present in the combined architecture. Note that the combined architecture $\lambda = 64$ is enough to maintain the accuracy and security of the sampler [7]. Therefore, in a resource constraint platform, one can use minimal parameters to reduce the area further. The Xilinx AXI IP will have additional cost in resource utilization based on the type used. The time required by the Rounder running on the PS (time-independent) to produce one sample is 43us. Furthermore, the permutation network used as the side channel countermeasure runs in constant time and returns shuffled batch indices in a single clock cycle. Note that since it does not lie in the critical path, it does not affect the final sample generation time.

TABLE II
Resource Utilisation of Design Modules

Design Block	LUT/FF/Slice	Delay (ns)
Base Sampler (c=0)	1241/0/579	24.26
Base Sampler (c=0.5)	1263/0/589	23.19
Shuffler (n=32)	773/0/170	3.06
ES-TRNG[30]	9/5/9	[a]

[a]the critical path varies depending on the sample

TABLE I
BASE SAMPLER AREA AND PERFORMANCE COMPARISON.

σ	Design	Device	λ	LUT/FF/Slice	BRAM	Clock cycle	Delay per sample (ns)
3.33	Howe et al. [20]	5VLX30-3	64	133/52/48	2	1.23	5.80
3.33	This work w/o shuffling	5VLX30-3	64	339/0/142	0	1	15.90
6.15543	Karmakar et al. (Batchinig) [10]	6VCX75T-2	112	1024/1237/113	15	27344	3204
6.15543	Karmakar et al. (Unrolled) [10]	6VCX75T-2	112	2682/977/*	*	1	4.9
6.15543	This work w/o shuffling	6VCX75T-2	112	1070/0/427	0	1	24.13

VI. CONCLUSION

We presented a compact and secure sampler implementation with 60% lesser LUTs than the previous architectures through multi-level logic optimization. The proposed sampler architecture is the first implementation in HW/SW co-design setting, generating a distribution with any arbitrary center and standard deviation. Moreover, we also present a lightweight random shuffling countermeasure to secure the sampler from side-channel attacks. The proposed scheme incorporates a permutation network with swap modules as building blocks. The permutation network is implemented on hardware, which executes in parallel without hampering the execution time. Therefore, such design choices and optimizations make this architecture/system a suitable source for Gaussian sample generation in lattice-based applications.

REFERENCES

[1] F. Arute, K. Arya, and et al., "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, no. 7779, pp. 505–510, Oct. 2019.

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

[3] J. Proos and C. Zalka, "Shor's discrete logarithm quantum algorithm for elliptic curves," *Quantum Inf. Comput.*, vol. 3, pp. 317–344, 2003.

[4] D. Micciancio and O. Regev, *Lattice-based Cryptography*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 147–191.

[5] O. Regev, "New lattice-based cryptographic constructions," *J. ACM*, vol. 51, no. 6, p. 899–942, Nov. 2004.

[6] M. Ajtai, "Generating hard instances of lattice problems (extended abstract)," in *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, ser. STOC '96. New York, NY, USA: Association for Computing Machinery, 1996, p. 99–108.

[7] M. D. and W. M., "Gaussian Sampling over the Integers: Efficient, Generic, constant-time," *Advances in Cryptology - CRYPTO. Lecture Notes in Computer Science*, vol. 10402, 2017.

[8] S. Sinha Roy, F. Vercauteren, and I. Verbauwhede, "High precision discrete gaussian sampling on fpgas," in *Revised Selected Papers on Selected Areas in Cryptography – SAC 2013 - Volume 8282*. Berlin, Heidelberg: Springer-Verlag, 2013, p. 383–401.

[9] S. S. Roy, O. Reparaz, F. Vercauteren, and I. Verbauwhede, "Compact and side channel secure discrete gaussian sampling," Cryptology ePrint Archive, Report 2014/591, 2014.

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

[11] A. Karmakar, S. S. Roy, F. Vercauteren, and I. Verbauwhede, "Pushing the speed limit of constant-time discrete gaussian sampling. a case study on the falcon signature scheme." in *2019 56th ACM/IEEE Design Automation Conference (DAC)*, 2019, pp. 1–6.

[12] D. Micciancio and C. Peikert, "Trapdoors for lattices: Simpler, tighter, faster, smaller," in *Advances in Cryptology – EUROCRYPT*, D. Pointcheval and T. Johansson, Eds. Springer Berlin Heidelberg, 2012, pp. 700–718.

[13] C. Peikert, *An Efficient and Parallel Gaussian Sampler for Lattices*. In: Advances in Cryptology-CRYPTO 2010, 2010.

[14] T. Pöppelmann, L. Ducas, and T. Güneysu, "Enhanced lattice-based signatures on reconfigurable hardware," in *Cryptographic Hardware and Embedded Systems – CHES 2014*, L. Batina and M. Robshaw, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 353–370.

[15] L. Ducas and P. Q. Nguyen, "Faster gaussian lattice sampling using lazy floating-point arithmetic," in *Advances in Cryptology – ASIACRYPT 2012*, X. Wang and K. Sako, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 415–432.

[16] 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," Cryptology ePrint Archive, Report 2013/510, 2013.

[17] L. Ducas, A. Durmus, T. Lepoint, and V. Lyubashevsky, "Lattice signatures and bimodal gaussians," in *Advances in Cryptology – CRYPTO 2013*, R. Canetti and J. A. Garay, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 40–56.

[18] C. F. F. Karney, "Sampling exactly from the normal distribution," *ACM Trans. Math. Softw.*, vol. 42, no. 1, Jan. 2016.

[19] D. Knuth and A. Yao, *The complexity of nonuniform random number generation, in Algorithms and Complexity: New Directions and Recent Results*. Cambridge, MA, USA: Academic Press 1976, 1976.

[20] 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, 2018.

[21] N. C. Dwarakanath and S. D. Galbraith, "Sampling from discrete gaussians for lattice-based cryptography on a constrained device," *Applicable Algebra in Engineering, Communication and Computing*, vol. 25, no. 3, pp. 159–180, Jun 2014.

[22] R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. L. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*. Springer US, 1984.

[23] M. Fujita, "Basic and advanced researches in logic synthesis and their industrial contributions," in *Proceedings of the 2019 International Symposium on Physical Design*, ser. ISPD '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 109–116.

[24] R. Brayton and A. Mishchenko, "Abc: An academic industrial-strength verification tool," in *Computer Aided Verification*, T. Touili, B. Cook, and P. Jackson, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 24–40.

[25] R. A. Fisher and F. Yates, *: Statistical tables for biological, agricultural and medical research*. Oliver and Boyd, 1948.

[26] M.-J. O. Saarinen, "Arithmetic coding and blinding countermeasures for lattice signatures," *Journal of Cryptographic Engineering*, vol. 8, no. 1, pp. 71–84, Jan. 2017.

[27] P. Pessl, "Analyzing the shuffling side-channel countermeasure for lattice-based signatures," Cryptology ePrint Archive, 2017.

[28] A. G. Bayrak, N. Velickovic, P. Ienne, and W. Burleson, "An architecture-independent instruction shuffler to protect against side-channel attacks," *ACM Trans. Archit. Code Optim.*, Jan. 2012.

[29] T. Schneider, C. Paglialonga, T. Oder, and T. Güneysu, "Efficiently masking binomial sampling at arbitrary orders for lattice-based crypto," in *Public-Key Cryptography – PKC 2019*, D. Lin and K. Sako, Eds. Cham: Springer International Publishing, 2019, pp. 534–564.

[30] B. Yang, V. Rožic, M. Grujic, N. Mentens, and I. Verbauwhede, "Esstrng: A high-throughput, low-area true random number generator based on edge sampling," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. Volume 2018, pp. Issue 3–, 2018.