# Primitives of Homomorphic Encryption Scheme: A Design Perspective

Project (EC57004) report submitted to

Indian Institute of Technology Kharagpur

in partial fulfilment for the award of the degree of

Master of Technology

in

Electronics and Electrical Communication Engineering


**by**

**Sudarshan Sharma**

**(15EC32005)**


**Under the supervision of**

**Dr. Debdeep Mukhopadhyay and Dr. Indrajit Chakrabarti**



**Department of Electronics and Electrical Communication Engineering1**

**Indian Institute of Technology Kharagpur**


**Spring Semester, 2019-20**

**June 5, 2020**

# DECLARATION

I certify that

(a) The work contained in this report has been done by me under the guidance of my supervisor.

(b) The work has not been submitted to any other Institute for any degree or diploma.

(c) I have conformed to the norms and guidelines given in the Ethical Code of Conduct of the Institute.

(d) Whenever I have used materials (data, theoretical analysis, figures, and text) from other sources, I have given due credit to them by citing them in the text of the thesis and giving their details in the references. Further, I have taken permission from the copyright owners of the sources, whenever necessary.

Date: June 5, 2020 (Sudarshan Sharma)

Place: Kharagpur (15EC32005)


## *CERTIFICATE*

This is to certify that the project report entitled "**Primitives of Homomorphic Encryption Scheme: A Design Perspective**" submitted by **Sudarshan Sharma** (Roll No. 15EC32005) to Indian Institute of Technology Kharagpur towards partial fulfilment of requirements for the award of degree of Master of Technology in Electronics and Electrical Communication Engineering is a record of bona fide work carried out by him under my supervision and guidance during Spring Semester, 2019-20.

Date: June 5, 2020

Place: Kharagpur

Dr. Debdeep Mukhopadhyay
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur
Kharagpur - 721302, India

Date: June 5, 2020

Place: Kharagpur

Dr. Indrajit Chakrabarti
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology Kharagpur
Kharagpur - 721302, India

# *Abstract*

---

Name of the student: **Sudarshan Sharma** Roll No: **15EC32005**

Degree for which submitted: **Master of Technology**

Department: **Department of Electronics and Electrical Communication Engineering**

Thesis title: **Primitives of Homomorphic Encryption Scheme: A Design Perspective**

Thesis supervisor: **Dr. Debdeep Mukhopadhyay and Dr. Indrajit Chakrabarti**

Month and year of thesis submission: **June 5, 2020**

---

After the emergence of cloud-based computation, Homomorphic Encryption Schemes has gained significant interest. This encryption scheme is based on Lattice-Based hard problems breaking those are considered difficult even the quantum computers till date, thus making them quantum secure. They also serve a suitable replacement of the existing Public Key Cryptography protocols like RSA and ECC, which are not considered quantum secure. The thesis presents the design perspective of the key elements required in the Homomorphic Encryption. The implementations are first simualated on C++, designed using Verilog, extensively simulated for different corner cases, then tested on actual FPGA Hardware and at the end, finally the outputs are also verified.

# *Acknowledgements*

I would like to take the opportunity to thank my thesis advisors Prof. Debdeep Mukhopadhyay and Prof. Indrajit Chakrabarti for giving me an chance to work under their guidance. I am greatly indebted to them for their invaluable guidance and support.

Furthermore, I would like to thank Mr. Arnab Bag (PhD, Department of Computer Science and Engineering) and my parents who have consistently helped me to pursue research. Additionally, friends and fellow researcher in the domain have all helped me a lot of times and made this journey memorable with out of the box ideas, insightful advice and funny jokes.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Motivation

These days cloud service plays a crucial role in the day to day life. There is a whopping increase in the usage of Social Media, Internet Banking, Business solutions, and numerous cloud services [Jul14]. The security and privacy over the cloud are one of the prime concerns, as to get the desired computation, one needs to send data to the cloud server, and since the cloud cannot be trusted, the owner of the cloud can see, use and abuse the unencrypted data. Homomorphic Encryption (HE) is a cryptography protocol that can help mitigate the privacy issue over the cloud without hampering the convenience and benefits of the cloud. The computations processed on the cloud server are performed on encrypted data, and the result obtained in the form of encrypted data is forwarded to the user. The user can then decrypt the results through his.her private key. Some of the exciting applications of the HE are privacy-preserving services which perform analytics on data collected to enhance the functionality and health-care applications [JWBN14], search engines with encrypted schemes [NPC13], electronic voting, and seclusion-preserving prediction from consumption data in future electricity meters [BCIV17], machine learning on encrypted data [GLN13] etc. Moreover, the HE schemes are based on learning with error hard problems, which are till date proved to be quantum secure. Thus, HE is a very suitable alternative for present public-key cryptographic algorithms like RSA, Elliptic Curve Cryptography (ECC). Based on the factoring problem, there is

an average-case hardness. On the other hand, HE schemes are based on worst-case hardness and are based on lattices.

## 1.2 Summary of the thesis

The thesis presentation is organized into four main chapters, excluding the Introduction and the Conclusion. The brief overview of the contents in the chapters are as follows:

- Chapter 2 gives a brief overview of the Homomorphic Encryption followed by a description of elements of Learning with Error Problem and its building block.
- Chapter 3 presents the Random Bit Generator architecture design on FPGA and statistical test results.
- Chapter 4 explains about the Gaussian Sampler algorithms and architectures. This chapter is the main point of interest of the thesis with extensive literature review, simulation, and results.
- Chapter 5 builds up the RLWE problem from the lattices. Moreover, it also describes the NTT from DFT. This chapter mainly includes the review of the present state of the art.

# Chapter 2

# Homomorphic Encryption : Brief Overview

## 2.1 Homomorphic Encryption

The secure outsourcing of computation or, in simple terms, a delegation of tasks to a cloud server through a secure protocol defines Homomorphic Encryption (HE) schemes. Initial schemes allowed only one operation. This is what we call partially (HE), the cryptosystem RSA [RSA78] is an example of partial HE. It is homomorphic over multiplication. The two RSA ciphertexts can be multiplied together and result in encryption of the product of their respective plain texts. Thus it is said that RSA is homomorphic for multiplication. The (HE) Schemes can be divided into two types, Somewhat Homomorphic Encryption (SHE) schemes which permit a number of operations known in advance and Fully Homomorphic Encryption (FHE) first step by [Gen09], where no more constraints pertain to the computation. Thanks to the novel technique of bootstrapping, which consists of homomorphically decrypting the ciphertext to refresh it, it becomes possible to evaluate any circuits on encrypted data. However, after a series of optimizations and new Homomorphic Schemes implementations are gaining much attention[Bon18]. In the subsequent year's modulus, switching was proposed by [Bra11] in 2011, which lower down the noise inherent in the ciphertext, which increases in each operation and thus postpones the usage of the bootstrapping. The second generation of(HE) Scheme was BGV [ZBV12], and BV [Bra11], Gentry's scheme is considered as the first one. SHIELD [KGV16] is the

only attractive third-generation scheme. It incorporates matrices compared to the second generation schemes, which needed only vectors. Therefore the minimal cost is higher, but as it does not require re-linearisation, the cost increases far slower than for previous generations.

The actual hardware implementation of the HE scheme requires a variety of additional building blocks to perform important functions, including memory management, parallel synchronization of cores, and reliable interfacing with a host processor, etc. [SSRV19]. This makes the implementation of complex HE schemes in hardware very challenging. The key elements/block essential for the design are as follows:

### 2.1.1 Learning with Errors (LWE)

Preferably Ring Based Learning with Errors (RLWE). As the former is computationally efficient due to polynomial based compared to matrix-based LWE. RLWE uses ideal lattices, Let Rq be the ring of polynomials. All coefficient of the polynomial was reduced to mod q. According to RLWE: samples of the form (a, a · s + e), it is difficult to determine the secret polynomial $s \in \mathbb{R}_{ıı}$, where the polynomial $a \in \mathbb{R}_{ıı}$ is sampled uniformly at random, and the coefficients of the error polynomial e are small samples from the error distribution $\xi$. [BUC19] The complex mathematical concepts and hardness of the scheme are proved in the thesis [Ber16]. The hardware and software architecture of the lattice-based cryptography has been thoroughly surveyed in [NDR+19]. The RLWE consists of the following elements, considering the design perspective.

#### 2.1.1.1 Discrete Gaussian Sampler

The Gaussian sampler is used to generate the error polynomial, which is used in the RLWE scheme. Presently, there exist many versions of distribution samplers for the RLWE scheme, but only the distribution changes, which means the probabilities. However, the algorithm remains the same as in the sampler used in TFHE[CGGI19].

#### 2.1.1.2 Modular Arithmetic core

The RLWE cryptosystem uses the polynomial arithmetic unit, which includes polynomial addition, subtraction, and multiplication. Polynomial addition and subtraction can be performed in the linear time only through coefficient-wise additions or subtractions modulo q. Computation of polynomial multiplication is the expensive operation in the RLWE based cryptographic schemes. Number Theoretic Transform (NTT) is considered a suitable option over FFT. Modular reduction post multiplication of polynomial is done either through the Barrett reduction or the Montgomery reduction.

#### 2.1.1.3 Number Theoretic Transform

The Number Theoretic Transform (NTT) corresponds to a Finite Fourier Transform (FFT) and is considered better than FFT because FFT and inverse-FFT perform arithmetic using real numbers and thus suffer approximation errors which are not desired in cryptographic applications. In NTT, the roots of unity are taken from a finite ring $Z_q$. Hence all computations in NTT are performed on integers. The NTT provides constraints on the parameters of the HE schemes one such condition says if and only if n divides $d1$ for every prime divisor d of q. The multiplication of the polynomial is done in $R_q$. There exist different iterative and optimized designs for the NTT [LN16]. The parameter of the type of HE schemes governs the optimization involved in the NTT.

# Chapter 3

# Random Bit Generators

## 3.1 Random Bit Generators

Random Numbers Generator are the core of the Cryptography Applications, it refers to many applications such as simulation, numerical analysis, machine learning etc.The paper [MBO18] presents a survey on the hardware implementation of Random Number Generator on FPGA. The main difference with the earlier generators are the use of periodic and deterministic output as an operator or arithmetic functions. They are known in literature as "pseudorandom" or "quasirandom" number generators (PRNGs), while circuits that takes into account a physical source to produce randomness are called "true" random number generators (TRNGs). The entropy sources present in these kinds of TRNGs are either the electronic noise of embedded components or some environmental sensors (temperature, noise, etc). The different kinds of TRNGs present in the literature are as follows:

- Phase Locked Loop (PLL) TRNGs. [FD03]
- Ring Oscillator TRNGs [MBO18] [BYV18]
- Self-Timed Ring TRNG [CFAF13]
- Metastability TRNG [VHKK08]

Considering the ease in integration, reliability, area utilisation and the throughput of the TRNGs Ring Oscillator based TRNG' have a very high speed and a higher and

more robust entropy rate in comparison with previous proposals for digital random number generators [Gol06].

### 3.1.1 Golic's Ring Oscillator RNG

First to use new methods and the corresponding robust techniques for generating high speed and high entropy raw binary sequence by using only logic gates in digital semiconductor technology. Then for the purely RNG output, additional post-processing of raw binary sequence that provide randomness extraction and computational security as well as increase in speed if required. The proposed method is independent of the fabrication technology.

#### 3.1.1.1 Random Binary Sequence Generation

The combination of the Fibonacci and Galois Ring Oscillators XORed together followed by the D Flip Flop clocked by a ring oscillator. In the Fibonacci Ring Oscillator, the inverters are used instead of the memory elements. The feedback connections are incorporated so as to follow the basic condition that there are no fixed points in the corresponding state-transition function. As seen from the Figure 3.1 it consists of r inverters connected in cascade, The feedback connections are specified by the binary co-efficient of $f_i's$ if $f_i = 0$ the switch is open is 1 it is in the closed configuration. The binary coefficients $f_i's$ are obtained from the co-efficient of feedback polynomial f(x). A proof is provided in the paper to get the essence of how the feedback polynomial is chosen. For the implementation purpose, the polynomial provided in the experiment section is used directly. The Galois ring oscillator also consists of r inverters in cascade with the $f_i's$ as the switches similar to the Fibonacci ring oscillator configuration. The randomness, as well as the robustness, is increased by XOR-ing the outputs of the two oscillators. The absolute value of the oscillators length minus one should preferably be mutually prime, first, to maximise the period of the corresponding pseudo random sequence and, second, to minimise the interlocking or coupling effect.

Because the oscillating signal is a mix of digital and analog noise, further randomness can be introduced using the meta stability of the sampling unit, e.g., implemented

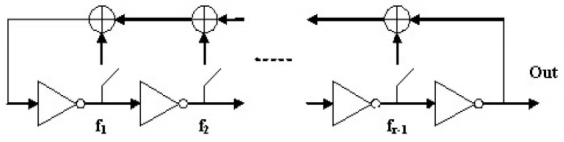


FIGURE 3.1: Fibonacci Oscillator. *Source: [Gol06]*

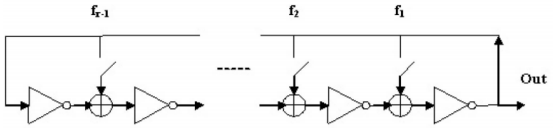


FIGURE 3.2: Galois Oscillator. *Source: [Gol06]*

as a D-type flip-flop. The clock signal used for the D flip flop is generated by a different ring oscillator with comparable frequency as the oscillating signal.

#### 3.1.1.2 The Post Processing Circuit

Once the design for generating the raw binary sequence is done there is some sort of bias still present which is removed using the post-processing circuit (Figure 3.3) present in cascade with the binary sequence generator. It is a self clock-controlled Linear Shift Feedback Register (LSFR) in Galois configuration similar to the oscillator above with the only difference being the delay elements as flip flops instead of the inverters. The output is formed by XOR-ing the Output of the XOR gate connected to the output of the $\gamma$th flipflop, where $f_{-1} = 1$. The output of the 2 input OR gate applied to the outputs of the $j^{th}$ and $k^{th}$ flipflops. The clock control signal is binary, so each time the output bit is produced, the value of the clock control bit decides the number of times the LSFR should be clocked 1 or 2. The clock control is implemented in hardware by a counting logic and another D-type flip-flop by using its clock enable input. The clock control bit is generated by XORing the outputs of the 1th and 2th flip flops as shown in the Figure 3.3.

#### 3.1.1.3 Hardware Implementation of Golic's TRNG Scheme

It is evident from the description of the circuit that the algorithm contains a lot of constants which needs to be calculated before the actual implementation. The

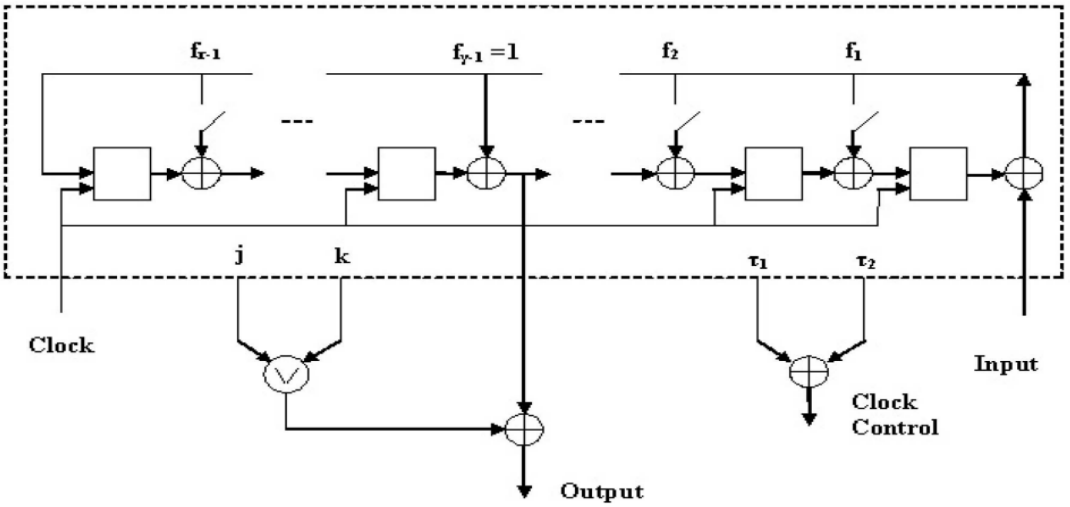


FIGURE 3.3: The Post Processing Circuit. *Source: [Gol06]*

number of the inverters in the fibonacci ring oscillator=20 in the Galois ring oscillator=21. The length of the LSFR taken as 64, the non zero co-efficient of the primitive polynomials are 64, 4, 3, 1, 0. The constants $\gamma 1 = 4$, $\gamma = 5$, j=20, k=25, $\tau 1 = 3$ , $\tau 2 = 2$ , $\delta 1 = 3$ and $\delta 1 = 5$. The hardware design was done on Vivado 2019.1 using the primitive FPGA Fabrics as the synthesis tool always optimised the ring oscillator redundant circuits even after using constraints. The design details is as follows:

TABLE 3.1: Golic's RNG Implementation Details

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs | 21 | 0 | 20800 | 0.10 |
| LUT as Logic | 19 | 0 | 20800 | 0.09 |
| LUT as Memory | 2 | 0 | 9600 | 0.02 |
| LUT as Shift Reg | 2 | 0 | | |
| Slice Register | 32 | 0 | 41600 | 0.08 |
| Register as Flip Flop | 32 | 0 | 41600 | 0.08 |

The FPGA used was Artix 7 Digillent FPGA (xC7A35TICSG324-1L).

#### 3.1.1.4 Issues with Golic's RNG Scheme

A greater number of constants involved. The selection of the primitive polynomial, the security is affected by the number of non-zero feedback coefficient. It is difficult to get the primitive polynomial of higher order. Absence of master clock in the design Only the long chain of oscillator act as clock for the output. After testing

the design, It can be concluded that the paper is quite old so maybe the LUT-LUT delay would have been greater compared to the present counterparts.

### 3.1.2 ES-TRNG

[BYV18] presents a lightweight, AIS-31 compliant TRNG with conservative entropy estimation. Compared to the existing TRNG architectures this work an additional emphasis is done on the stochastic model based formal security analysis.Edge Sampling, variable-precision phase encoding and repetitive sampling are used. Edge Sampling is an algorithm in which we select N edges at random from a graph. Nodes connected to those edges are present in the output network. Variable-precision phase encoding is the encoding method in which Only the oscillators phase around the single edges are sampled with higher precision. Repetitive Sampling means the repetition of the sampling at higher frequency until the high precision phase region of the oscillator is captured. Proposed scheme has a throughput of 1.15 Mbps with 0.997 bits of Shannon entropy, only 10 LUT and 5 flip-flops are used on Spartan-6 FPGA (fairly lightweight). The design criterion considered are resource consumption, throughput, latency, feasibility, design efforts, resistance against attacks and a stochastic model to prove unpredictability and inner test ability of the entropy source. Most of the existing TRNS's are tested with [RV01] NIST Test, it is said that for the NIST tests the statistical evaluation of the output is bad because a completely deterministic pseudo-random sequence could pass the test despite having no randomness. Thus, modern approaches like the BSI AIS-31 uses a stochastical model to evaluate the unpredictability.The unpredictability of the TRNG depends on some physical processes here timing phase jitter in a free running ring oscillator is leveraged to generate a digital noise source. Raw Random Numbers obtained directly from the digital noise source suffer from statistical defects bias, ideal probability of ones auto-correlation between output bits. Post Processing helps to mitigate the above-mentioned error, algorithmic post-processing extracts entropy to from raw random numbers to increase entropy per bit. Furthermore, cryptographic post-processing helps in backtracking resistance [Yan18]. The Post Process sing is performed by XORing the three consecutive raw bit generated from the digital noise source.

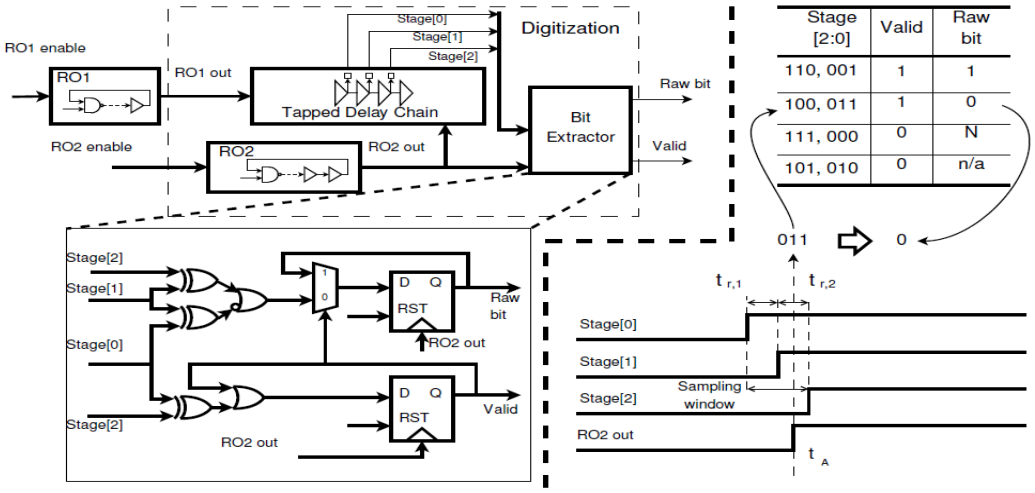


FIGURE 3.4: The Digital Noise Source Circuit. *Source: [BYV18]*

#### 3.1.2.1 Digital Noise Source

The entropy source denoted by RO1is implemented as a free-running ring oscillator with an enable signal. The average period of RO1is denoted by T01. The output signal of RO1 propagates through the Digitization module.The digitization module consists of three main components, namely, a tapped delay chain, a sampling free-running ring oscillator RO2 and a bit extractor. The average period of RO2 is denoted by T02. The used tapped delay chain consists of four cascaded delay elements, the first and last one acting providing isolation for the middle two delay chains. The output from the tapped delay Stage [2:0] chain is processed by the combinatorial circuit of the bit extractor which encodes these values to a single raw bit which is the output of the digital noise source and a strobe signal Valid which is responsible to disable the ROs for the new set of random bit generation.

#### 3.1.2.2 Variable Precision Encoding

This technique Figure 3.5 is enabled by using both the tapped delay chain and the bit extractor. The oscillators phase around the single edge is sampled with higher precision and the remaining region's phase with lower precision. The position of the captured edge is encoded into a raw bit. This result in compact implementation, shorter jitter accumulation time and entropy is reduced since the low precision is used.

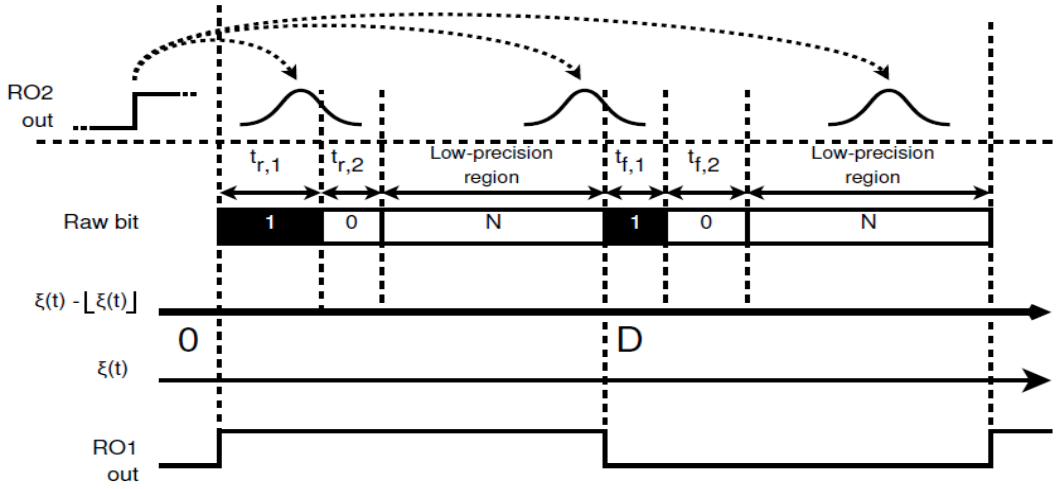


FIGURE 3.5: Variable Precision Encoding Scheme. *Source: [BYV18]*

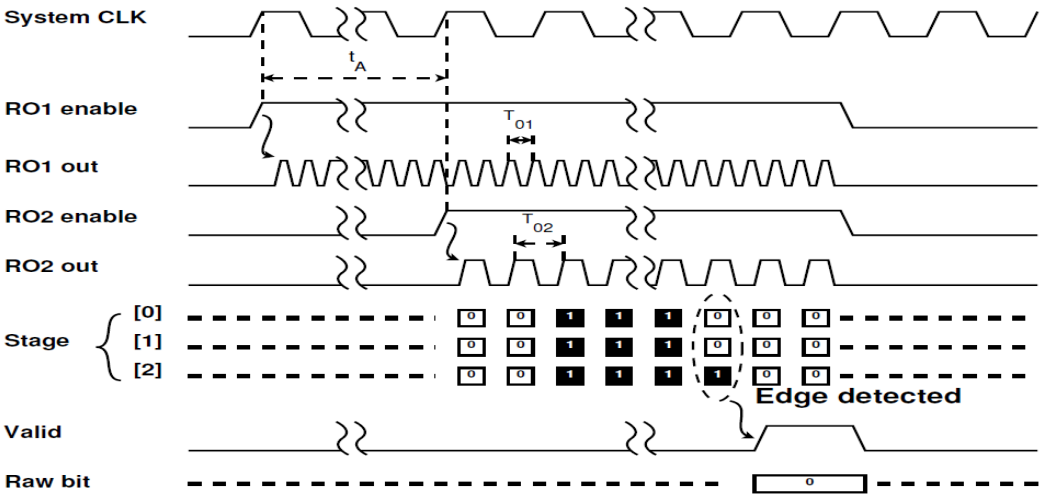


FIGURE 3.6: Waveform for Repetitive Sampling. *Source: [BYV18]*

#### 3.1.2.3 Repetitive Sampling

Repetitive sampling Figure 3.6 is synchronized to the high frequency signal RO2 out, aiming to reduce the time needed to hit the high-precision region, thereby improving the throughput. Once the high-precision region is hit, a Valid signal is generated. This helps in improving the throughput. The jitter accumulation time tA is chosen to allow accumulating enough jitter at the entropy source. This sampling multiple times within a single system clock cycle, because the frequency of a free-running RO2 is higher than the system clock's.

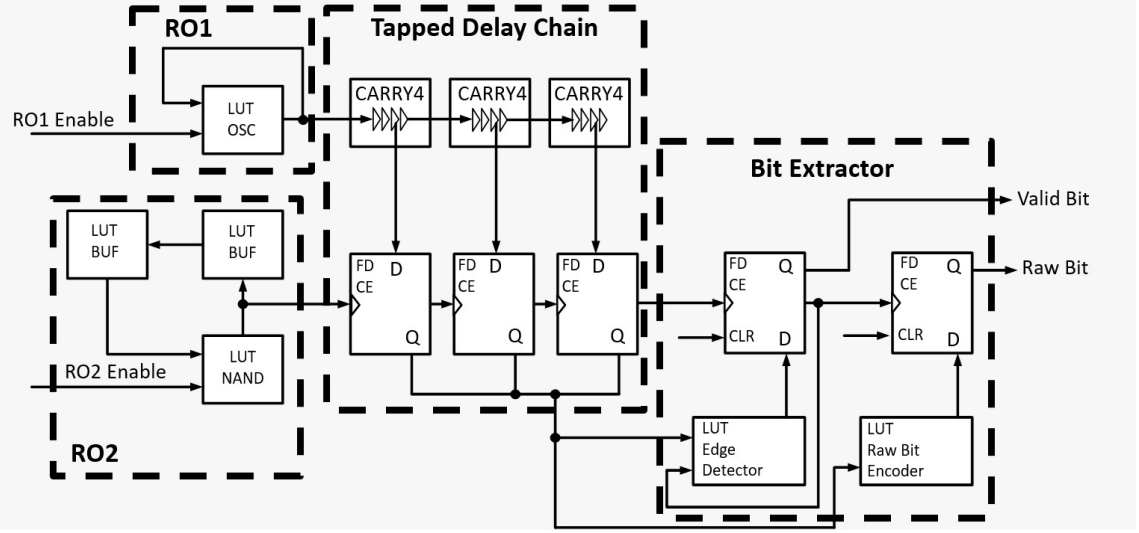


FIGURE 3.7: FPGA Architecture

#### 3.1.2.4 Hardware Implementation of ES-TRNG

Since, the TRNG is very compact in terms of the implementation it can be easily implemented using the FPGA Fabrics. The author presents the design on Spartan 6 with the Hardware Architecture as seen in Figure 3.7. The tapped delay chain implemented using the carry chain and flip flops, the input of the flip flop comes from the XOR gate of the carry chain. The tapped delay chain as in Figure **??**, the $\tau$ represent the delay due to the multiplexer and the XOR gate delay in the carry chain.

TABLE 3.2: ES-TRNG Implementation Details

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs | 9 | 0 | 20800 | 0.04 |
| LUT as Logic | 9 | 0 | 20800 | 0.04 |
| LUT as Memory | 0 | 0 | 9600 | 0.00 |
| Slice Register | 5 | 0 | 41600 | 0.01 |
| Register as Flip Flop | 5 | 0 | 41600 | 0.01 |

The FPGA used was Artix 7 Digillent FPGA (xC7A35TICSG324-1L) and the tool as Vivado 2019.1.

### 3.1.3 Statistical Tests Results

| NIST Statistical Test | Pass Rate | P Value | Result |
|---|---|---|---|
| *Frequency* | 2/10 | 0.000000 | FAIL |
| *Block Frequency* | 10/10 | 0.122325 | PASS |
| *Cumulative Sums* | 2/10 | 0.000000 | FAIL |
| *Runs* | 3/10 | 0.000000 | FAIL |
| *Longest Run* | 7/10 | 0.534146 | FAIL |
| *Rank* | 10/10 | 0.122325 | PASS |
| *FFT* | 10/10 | 0.739918 | PASS |
| *Non Overlapping Template* | 8/10 | 0.017912 | PASS |
| *Overlapping Template* | 10/10 | 0.739918 | PASS |
| *Universal* | - | - | - |
| *Approximate Entropy* | - | - | - |
| *Random Excursions* | - | - | - |
| *Random Excursions Variant* | - | - | - |
| *Serial* | 10/10 | 0.534146 | PASS |
| *Linear Complexity* | 9/10 | 0.004301 | PASS |

FIGURE 3.8: NIST Test Results

Both the NIST SP800-22 as well as the AIS- 20/31 tests were performed on approximately 6 million bits ontained from the design running on FPGA through the UART. Therefore, the NIST test was performed on 10 sequences each of size around 600,000 bits.

The pass rate for the NIST test for a set of 10 sequences is at least 8 with exception of the Random Excursions Test and the Random Excursions Variant Test. Also, it is to be noted that if a test passes less than the expected number of sequences, we get a 0.0000 as the P-value. Such is the case for Frequency, Cumulative Sums and Runs rows as seen in the Figure 3.8. In order to generate P-values for the Universal, Approximate Entropy, Random Excursions, Random Excursion bits, we need at least 100 million bits.

All the sub tests in the AIS-20/31 test passed for this collected data. The AIS-31 test is regarded better than the NIST test.

| AIS 20/31 Test | Result |
|---|---|
| **Procedure A** | |
| T0 | PASS |
| T1 | PASS |
| T2 | PASS |
| T3 | PASS |
| T4 | PASS |
| T5 | PASS |
| **Procedure B** | |
| T6 | PASS<br>d = 0.004530 < 0.025<br>s = 0.001690 < 0.02 |
| T7 | PASS<br>s1 = 0.3025961 < 15.13<br>s2 = 0.044185 < 15.13 |
| T8 | PASS<br>s = 8.108488 > 7.976 |

FIGURE 3.9: AIS Test Results

# Chapter 4

# Discrete Gaussian Sampler

## 4.1 Discrete Gaussian Sampler

The continuous Gaussian distribution with the mean $c \in \mathbb{R}$ and a standard deviation of $\sigma > 0$ is defined as follows. Given that E is a random variable on $\in \mathbb{R}$, then for $x \in \mathbb{R}$.

$$Pr(E = x) = \frac{e^{-(x-c)^2/2\sigma^2}}{\sigma\sqrt{2\pi}} \tag{4.1}$$

The discrete Gaussian distribution on $\in \mathbb{Z}$ with a standard deviation of $\sigma > 0$ and mean 0 is defined as follows. Given E as a random variable in $\in \mathbb{Z}$

$$Pr(E = z) = \frac{e^{-(z)^2/2\sigma^2}}{S} \tag{4.2}$$

S is termed as the normalization factor approximated as $\sigma\sqrt{2\pi}$; the value of the parameter S can be calculated by summing the probability from $-\infty$ to $\infty$. The constant S is used to define the distribution as if S knows the standard deviation can be calculated through the approximate relation mentioned.

$$S = 1 + 2\sum_{z=1}^{\infty} e^{-(z)^2/2\sigma^2} \tag{4.3}$$

The sampled discrete Gaussian distribution accuracy is measured using the statistical distance. The statistical distance between the theoretical discrete distribution and the sampled distribution is kept as small as possible to satisfy the security proof of the cryptography protocols [Fol15]. It is well known that the range of the Gaussian is from $(-\infty, \infty)$. In the practical scenario, a tail-cut factor $\tau$ is used for the sample generation. The typical interval of the sample is $[-\tau\sigma, \tau\sigma]$ ignoring the other values. This is also termed as tail bound in some cases. Another important metric involved in the discrete Gaussian distribution is the precision of the probability distribution denoted as $\lambda$. It is also called the security parameter, as it constitutes an important role in the statistical distribution calculation.

### 4.1.1 Existing Methods

It is known that the tail bound of the Gaussian distribution is infinitely long; none of the sampling algorithms can over the entire tail bound. So, sampling is performed up to a tail bound. This tail bound is responsible for the statistical distribution. The tail bound is calculated according to the maximum statistical distribution allowed by the security parameter. Thus, the sampling methods need either a large number of floating-point operations or stored large precomputed tables. Due to this reason, the implementation of these samplers become difficult on constrained memory devices without high precision floating points arithmetic [Roy17].

The detailed survey on the various method of sampling discrete distribution is summarised in [DG14]. The two major sampling methods are rejection sampling and the inversion sampling method. In Rejection Sampling, a point is generated randomly, and then those points inside the distribution are accepted. This method is slow for Gaussian (High rejection rate) also many random bits required due to the high rejection rate for sampled values near the tail. The rejection sampling method doesn't use precomputed tables. However, because of the high rejection rate, the latency is hindered. A modified version of this algorithm, along with significant speedup and lower precision, is presented by [DN12]. Moreover, In the inversion method, a uniform random variable U in the interval [0, 1] is obtained, and the output Z is

computed according to the inequality. where $p_i$ is Pr(Z=i).

$$F(Z-1) = \sum_{i<Z} p_i < U \leq \sum_{i<Z+1} p_i = F(Z) \tag{4.4}$$

[Pei10] proposed to use this method to sample discrete Gaussian. He used precomputed tables to store the F(Z) values, and the inequality was solved performing a binary search in the table. This method has a higher throughput without using the high precision floating-point arithmetic, but it also requires the presence of precomputed tables, increasing memory utilization. These tables can be relatively large, depending on the $\sigma$ and the tail cut bound, and multiple such instances are required: one for each different parameter combination. When this algorithm is implemented in hardware due to the summation involved, the comparator size increases. Also, the random bits required from the uniform distribution needs to have high precision. Another type of Gaussian Sampler, which is also considered efficient, is known as the Cumulative distribution table (CDT) based sampling. In this algorithm, first, the cumulative distribution function table is precomputed. The sampling phase consists of a search operation on the table created according to the random value in the interval [0,1). This random value is then identified between the two consecutive table values, and the lower index among the consecutive index is returned as the output sample. The search efficiency is improved using a binary search with a guide table. This method is vulnerable to a side-channel attack as the table access pattern of binary search, which is irregular, can be used to mount a cache timing attack. The Knuth Yao sampler is chosen for the study because of its efficiency, low entropy consumption, and constant-time implementation in various platforms [HKR+18]. Moreover, the validation of choice is presented in detail in the following sections.

### 4.1.2 Knuth Yao Algorithm

The Knuth Yao Algorithm [KY76] uses a random walk method of the Discrete Distribution Generating (DDG) to perform sampling using the probabilities of the elements in the sample space.

#### 4.1.2.1 Features

The number of random bits required is very close to the entropy of distribution almost near-optimal. Knuth Yao proposed a random walk model for sampling from any non-uniform distribution. The algorithm can be implemented through a series of optimizations and is one of the states of art Gaussian sampler implementation on constrained devices.

#### 4.1.2.2 Method

The first implementation of the algorithm was given by [SRVV14]. The method is relatively simple. The sample-set say it contains m elements, and the $i$ element is sampled with the probability of $p_i$. So, convert the probability in binary and represent in the form of a DDG tree. The tree is traversed according to a random bit starting from the root; when the next node is the leaf node, the algorithm terminates, and the sampled value is given as output.

#### 4.1.2.3 DDG Tree

Discrete Distribution Generating Trees is a rooted binary tree. There are two types of nodes in the DDG tree.

- Intermediate Node
- Terminal Nodes

The intermediate nodes are responsible for the creation of the next set of levels in the DDG tree. It is obtained from the probability matrix of the sample set since the length of the sample set is m, the length of the number of rows in the probability matrix would be m. The number of columns say p of the probability matrix say $P_{mat}$ depends on the floating-point precision. The $i^{th}$ sample element will have the probability $p_i$.

**Property** The number of terminal nodes at a level say i in a DDG tree is given by the hamming weight of the $i^{th}$ column of the probability matrix.

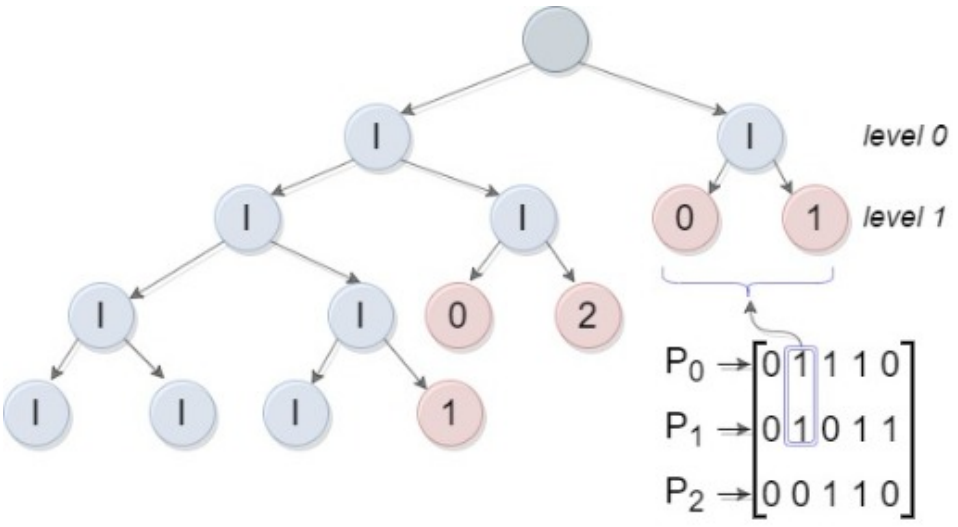


FIGURE 4.1: DDG Tree creation using the $P_{mat}$.

#### 4.1.2.4 DDG Traversal

The traversal stands with the root and depending on the random bit, a convention in chosen say if a random bit is 0 go to left and right in case random bit yields 1. Using this convention, the traversal continues until it hits a terminal node. The sampling process is terminated once a terminal node is hit, and the value at the node is returned. The node value is the output sample. There exists an exception case when the DDG traversal hits the additional appended row, which is 1- $P_{sum}$, as described in section 4.1.2.7. First of all, the probability of such an occurrence is very low, and it happens the sampling process is discarded for that particular instance, and the sample is generated again.

#### 4.1.2.5 DDG Creation

As described in the Figure 4.1, the root node of the DDG splits into two intermediate nodes as the first column of the $P_{mat}$ will always be 0 since the binary representation of probabilities will always be of the perform $0.b_{-1}b_{-2}.....b{-}p$ where p is the number of columns in $P_{mat}$. The first level of the DDG is constructed using the $1^{st}$ column of the $P_{mat}$. In the Figure 4.1 for the level 3, consider the $3^{rd}$ column, the hamming weight of the $3^{rd}$ is 1. So, the only terminal node in this level would be the row index one, which is written on the right node of level 3. The terminal node is marked with the row index, where one is present in the selected column vector. It is important

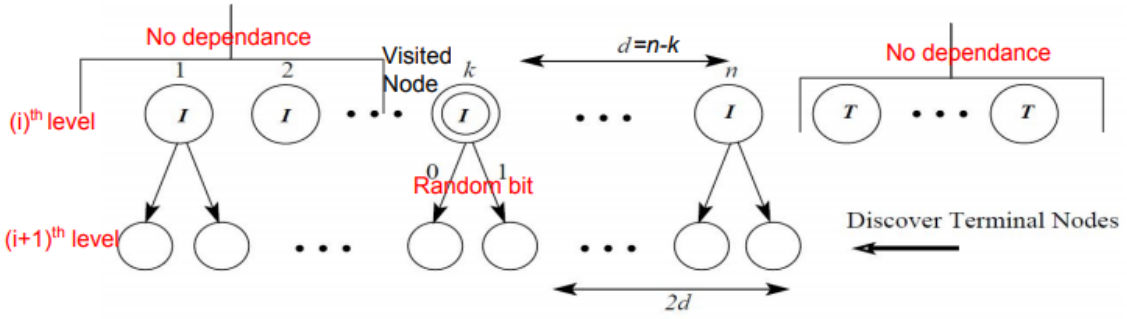


FIGURE 4.2: Optimisation of DDG Traversal. *Modified from Source: [SRVV14]*

to a node that the nodes at a level are filled in the right to left fashion; the row index searches for 1's in the selected column vector in a bottom-up fashion which means first the last row index is checked for one if found the index is noted in the rightmost node of the selected level.

The traversal starts from the root node; a random bit is required to predict the next node. If the random bit is say 0, the left node is selected and vice-versa. The sampling operation is terminated when it hits a terminal node; the terminal node gives the row index of the sample in the $P_{mat}$.

#### 4.1.2.6 Optimisation in DDG Traversal

In the hardware, it is not easy to create the entire DDG tree due to large memory overhead. The $(i+1)^{th}$ level of the DDG tree can be created from the $(i)^{th}$ level. According to the algorithm, the $i^{th}$ level of the DDG needs only the hamming weight of the $i^{th}$ column, and the algorithm terminates once it hits the terminal nodes. Figure 4.2 explains the traversal of the DDG tree; it can be observed that the visited node is a terminal node in the $i^{th}$ level. Now, since the hamming weight of $i^{th}$ level is known, the number of intermediate nodes is known as (Terminal + Intermediate = Total Nodes in the Level). The visited node k is d nodes to the right of the terminal node n, d=n-k. Now, we need to find whether we hit the terminal node or intermediate node in the $(i+1)^{th}$ level the number of terminal nodes would be on the right side which will be decided by the $(i+1)^{th}$ column's hamming weight. Depending on the random bit and the value of 2d, the decision can be inferred.

#### 4.1.2.7 Optimisation in $P_{mat}$ Storage

The probability matrix $P_{mat}$ can be stored efficiently; the storage depends on tail bound and precision. Data is stored in the RAM of the hardware, Data fetching from the RAM increases the computation time, which affects the performance of the algorithm. As $P_{mat}$ is accessed column-wise, store $P_{mat}$ columns in ROM Words. It can be observed that near the bottom of the column, a large number of zeros are present in the $P_{mat}$. The column length can be used to compress the zeros; it is the top portion's length, after which the chunk of bottom zeros starts. The differential column length, i.e., for the next column, store the difference in the column length of the present and next column. Figure 4.3 depicts the $P_{mat}$ conversion to compressed ROM words, here underscore used to separate the two columns. Also, the values 0 and 1 not included in the rectangle are the differential column length 0 means the same column length and 1 means increment column length by 1. If the sum of all the probabilities for a certain tail bound is not equal to 1. It is unsure whether the Knuth Yao Algorithm will hit a terminal node. To mitigate this issue, use one extra sample point valued as (1-$P_{sum}$), which, if it is discarded and the probability of such an event is very, very low. Hence, this sample point is regarded as out of range event.

### 4.1.3 Hardware Implementation of Knuth Yao Algorithm

Based on the previous sections' algorithm, the main components involved in the hardware architecture are as follows:

- ROM stores the probability matrix (compressed) the probabilities are accessed through ROM counter Initially, the counter is cleared and incremented one by one to fetch data from higher ROM locations.
- Data from the ROM is stored in a SCAN register, which is a left shift register A counter is used to count the number of bits scanned from the ROM when this value reaches the WORD SIZE of the ROM, the ROM counter is incremented, and Data is fetched from the ROM.
- A Random Bit generator is required to generate the random bit for DDG traversal. Random bit generator based on the approach of [Gol06] is used in

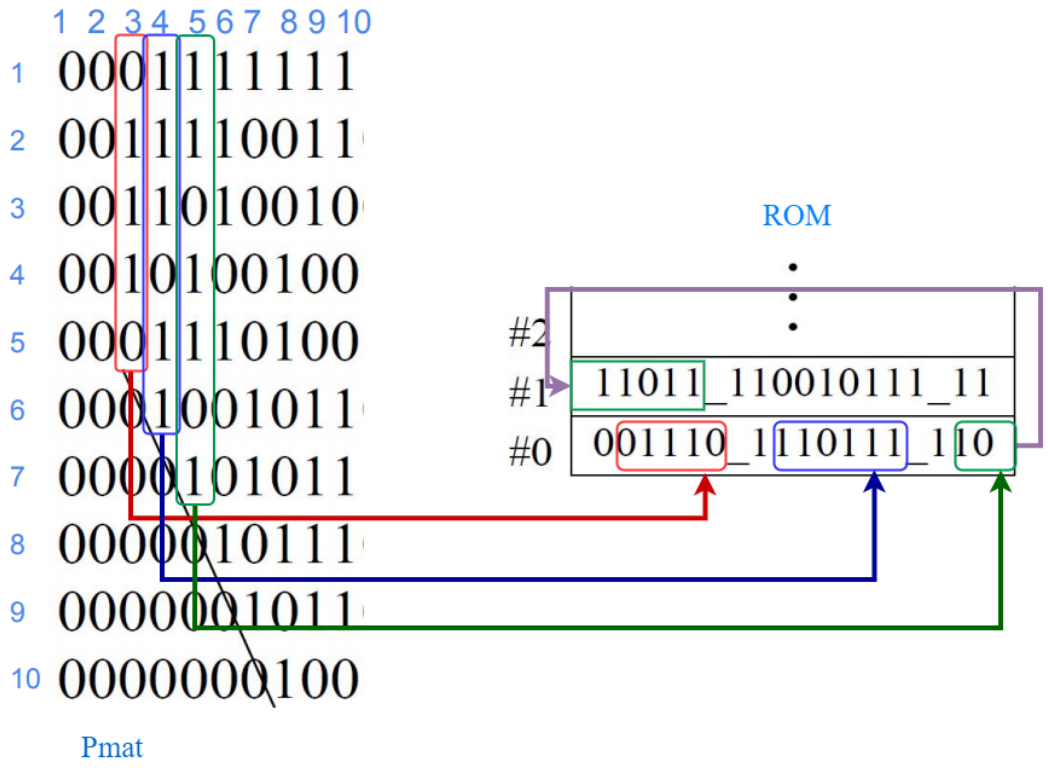


FIGURE 4.3: Optimisation of Matrix Storage. *Modified from Source: [SRVV14]*

the Paper Real or pseudorandom generators can be used. On average, only 5 bits are required. Thus the random bit generator can be slow.

- Upcounter column length stores the column lengths of different columns of $P_{mat}$. Initialised to first non-zero length of $P_{mat}$. During the Random walk, the counter remains as it is or increases by 1

- A Row Down counter, is used to count the number of rows in the $P_{mat}$ during column scanning, i.e, HW. This value is set to the Column length and then decremented one by one to reach zero.

- During the random walk, the "d" is stored in the distance register. It is updated to 2d or 2d+1 depending on the random bit. Later each detection of terminal node i.e., the 1 in the column decrements this distance register by 1. Any moment when d¡0. The terminal node is returned as the sample output, and the sampling process terminates.

- A Finite State Machines (FSM) Circuit is used to control the various multiplexers; counters present the architecture.

### 4.1.4 Knuth Yao Sampler C++ Implementation

**Algorithm 1:** Knuth Yao Algorithm Column Scanning [KRVV19]

**input**: Probability matrix P

**output**: Sample value s

*dist* = 0;// Distance between the visited and the rightmost internal node

*Hit_terminal* = 0;// 1 when sampling process hits a terminal node

*col_index* =0;// column number of probability matrix

**while** *Hit_terminal=0 and col_index* < *MAXCOL* **do**

    *r=get_RandomBit()*;

    *dist*=2*dist+r;

    **for** *row_index=MAXROW down to 0* **do**

        *dist*=dist-P[row_index][col_index];

        **if** *dist=-1* **then**

            *s*=row_index;

            *Hit_terminal*=1; *ExitForLoop()*;

        **else**

        **end**

    **end**

    *col_index*=col_index + 1;

**end**

**return** s

The random bit source for the simulation is the NTL C++ library[Sho09]. The Gaussian Sampler with the following parameter was simulated.

$\sigma = 6.33$

$\tau = 4$

center=0

$\lambda = 107$

iterations=1 million

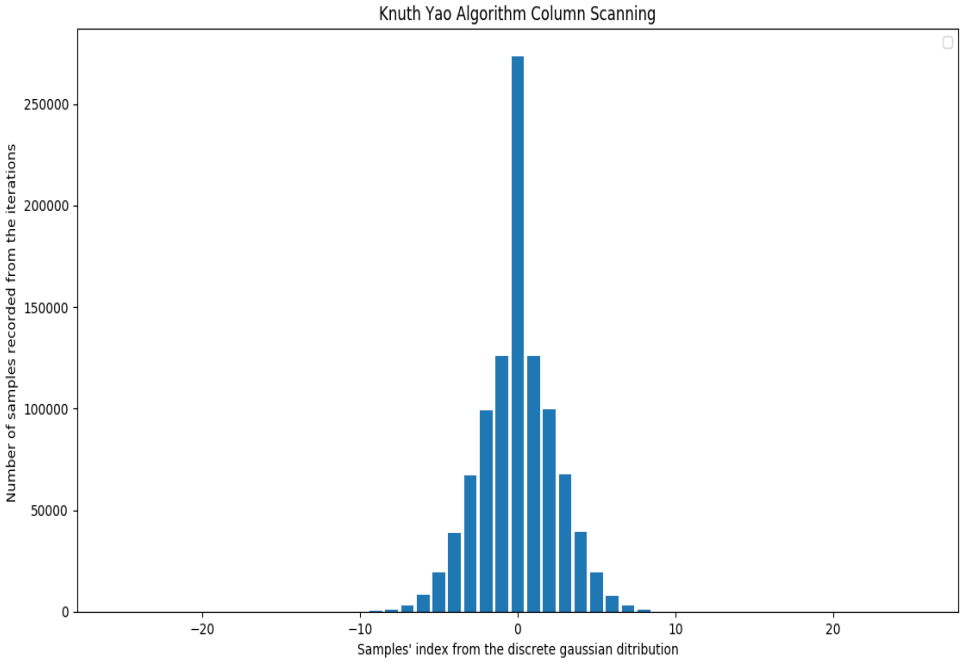


FIGURE 4.4: Knuth Yao Algorithm $\sigma$=6.33 Column Scanning.

$\sigma = 16$

$\tau = 4$

center=6

$\lambda = 107$

iterations=1 million

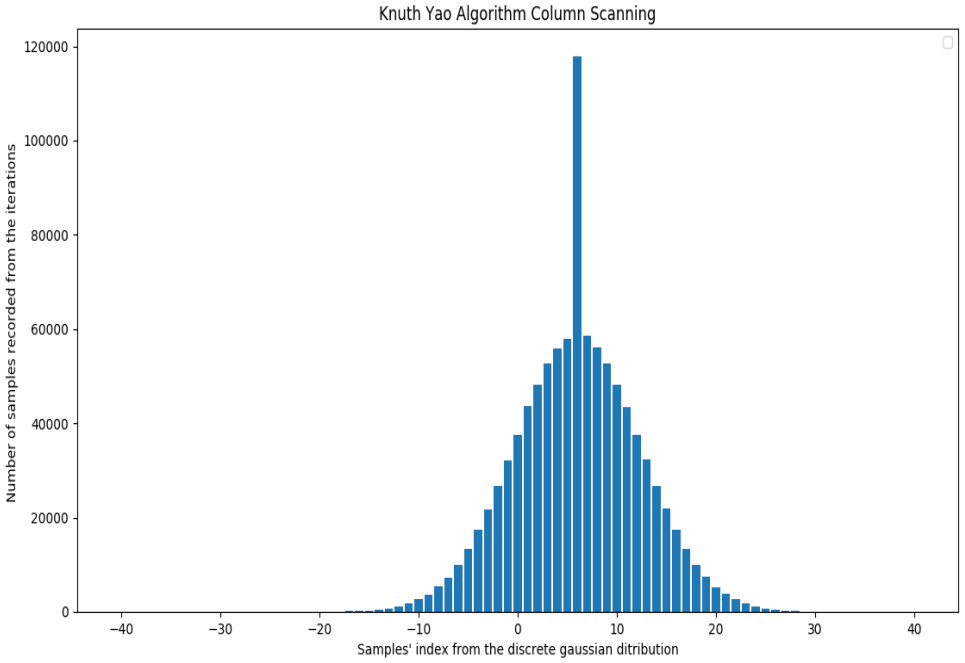


FIGURE 4.5: Knuth Yao Algorithm $\sigma$=16 Column Scanning.

### 4.1.5 Side Channel Attacks Review

The recent advancement and efficiency improvement of Lattice-based cryptography has made it a prime choice, which has led to its implementation on a variety of devices. The Gaussian Sampling being the most crucial block is vulnerable to side-channel attacks. The work by Bruinderink et al. [GBHLY16] uses the cache timing of the table access pattern of CDT based sampling to break the sampler. Moreover, Peter Pessl [Pes17] in his work attacked the popular shuffling based countermeasure for the Gaussian Sampling with a larger number of signature. Another kind of countermeasure involving Gaussian convolution in conjunction with shuffling twice was also attacked with a comparatively larger number of signatures. A lot of work has been done to make the Gaussian Sampler secure against the side-channel vulnerability. The following sections will describe all the methods proposed in this area.

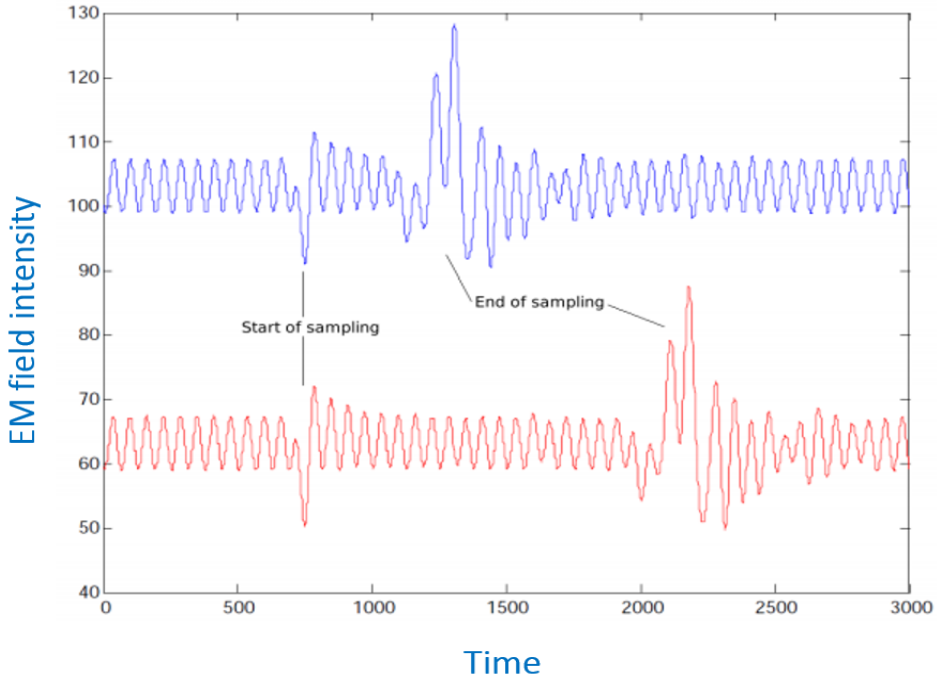


FIGURE 4.6: EM Signal Intensity measurement for two samples. *Source: [SSRV14]*

#### 4.1.5.1 Roy etàl [SSRV14] Compact and Side Channel Resistant Discrete Gaussian Sampling

The author first showed the vulnerability in the hardware implementation of the Gaussian Sampler using the Simple Power Analysis. It is clearly visible from Figure 4.6 that it is easy for an attacker to guess the sample value depending on the number of cycles it takes for the sampling operation. Intuitively, it can be observed that every sample in the DDG tree cannot be obtained with the same number of random bits. In other words, the terminal nodes of the DDG tree are not equidistant from the root. To prevent such attacks, the need to be implemented in constant time. The authors propose a two-part sampling approach to achieve constant-time implementation. This method initially stores the first k columns of the probability matrix in a table with $2^k$ entries. The entries are either a sample value or an intermediate position in the DDG tree. Furthermore, the sampling is divided into Secure and Non Secure sampling. As the name suggests in the case of Secure sampling, the sampler doesn't leak any information about the sample; in part, the k bit generated random bit is used for table lookup, which returns a sample value. Now, if this table lookup returns

an intermediate position, then the random walk of the DDG from that position is initiated to find the sample. It is trivial that the latter will leak information about the sample due to the difference in sampling time, and hence this part is termed as Non Secure sampling. To prevent the information leakage in the Non Secure part, a second countermeasure is proposed, which involves a shuffling technique. This shuffling technique is a random permutation of the leaked and non-leaked sample to prevent the sampler from extracting the location of the samples. The random shuffling method incorporated is the modified version of the Fisher and Yates shuffle, also termed as Knuth shuffle. The key bottlenecks are the exponential memory requirement with an increase in security level and the increase in the implementation area due to additional random shuffling operation.

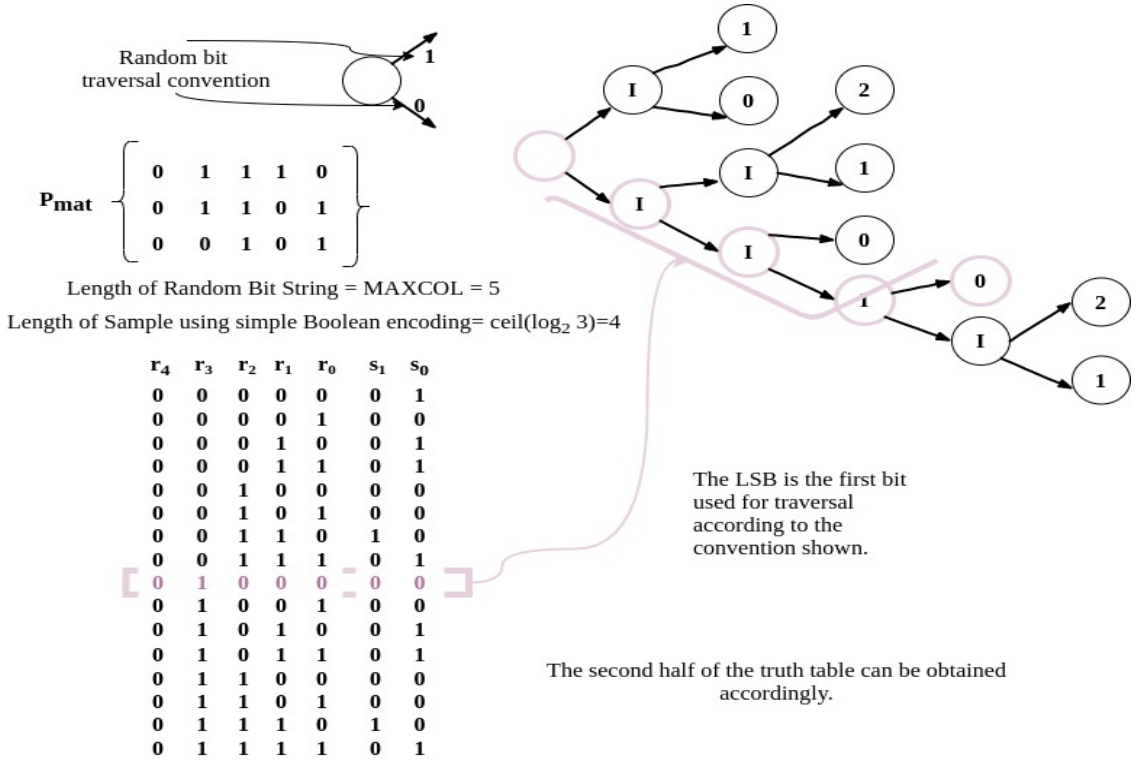


FIGURE 4.7: Truth Table from DDG

#### 4.1.5.2 Karmakar et.al [KRR+18] Constant-time Discrete Gaussian Sampling

In this follow up work by the same research group (Roy et al. work). The authors present a constant time Gaussian Sampler avoiding the data-dependent branching

present in the tree traversal of the Knuth Yao Algorithm. This is achieved by obtaining a unique map between the output sample value and the input random bit string used in the algorithm. The output samples are then expressed in the form of boolean expression in terms of the input random bit string. The constant time is attained through the evaluation of these boolean functions in constant time. Further, the author's sample in batches which increases the throughput of the sampler using the bit slicing technique which uses the Single Instruction Multiple Data (SIMD) approach by exploiting the data level parallelism. The authors also evaluate the implementation and compare it with the present available alternatives implementations. The key bottleneck involved in this method is the fact that it requires larger program memory to store the boolean expression of the mapping. Also, it can be observed that all the output samples doesn't require the same number of input random bits. This can be understood by observing the DDG tree in which not all output samples are at the same depth compared to the root node. Thus, some of the input random bits are wasted for some of the samples.

#### 4.1.5.3 Karmakar et al. [KRVV19]Pushing the Speed Limit of Constant-time Discrete Gaussian Sampling. A Case Study on the Falcon Signature Scheme

| $r_4$ | $r_3$ | $r_2$ | $r_1$ | $r_0$ | $s_1$ | $s_0$ |
|---|---|---|---|---|---|---|
| x | x | x | 0 | 1 | 0 | 0 |
| x | x | 1 | 0 | 0 | 0 | 0 |
| x | 1 | 0 | 0 | 0 | 0 | 0 |
| x | x | x | 1 | 1 | 0 | 1 |
| x | x | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| x | x | 1 | 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 | 0 |

FIGURE 4.8: Modified Truth Table from DDG.

This work extends the previous work of Karmakar et al. [KRR+18]. The author exploits the unique mapping of the input random bit string and the output sample mapping observing the unique property of the input random bit string. According to the property the all the input random bit string 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. This property takes into consideration that not all random bits are used for some output samples. The constant time implementation is governed by the constant time execution of the if/else blocks used in the bit-slice implementation of DES in [Bih97]. The efficient optimization of the boolean expression is done by exploiting the observed property, based on the value of k, the input random bit strings and the output samples are separated in different lists. Theses lists are then sorted based on increasing order of k. Now, these sorted lists are implemented in the if/else fashion by strategically using the 0 present in each of the lists just after $1^k$ towards the left. The authors observe a throughput increase of 37% after this efficient minimization technique. The key bottleneck involved in this method is it requires large program memory for larger standard deviations and has lesser modularity. Moreover, the author doesn't mention the hardware implementation and associated optimization.

### 4.1.6 Combining Samples

It is observed that the increase in the standard deviation of the Gaussian Sampler increases the memory requirement as well. For a resource-constrained environment, say an IoT module it is challenging to implement a Gaussian Sampler with $\sigma = 215$ typically used in BLISS [DL13] signature scheme. The idea of combining samples of smaller standard deviation in an iterative fashion to obtain the required distribution was first proposed by P "oppelmann et al. [PDG14]. The authors here used the Kullback-Liebler divergence to show the closeness similar to the statistical distance. Further, Micciancio at.al [MW17] generalized this idea and formulated a constant time robust method to generate samples from Gaussian distribution with any center and standard deviation on the fly. Moreover, they also formulated a new metric to measure the closeness of the distribution termed as the max-log distance. The proposed Gaussian Sampler builds upon this idea and discusses more on the hardware level architecture in the following sections.

### 4.1.7 Proposed Gaussian Sampler

The majority of content in this section has a significant overlap with the work by Micciancio et al. [MW17] and Karmakar et al. [KRVV19]. It is evident from the previous discussion that for Gaussian Sampler with higher standard deviation, intuitive choice suggests the recombination of samples. Further, to achieve constant-time implementation to mitigate the side-channel vulnerability with minimum performance and implementation cost, a robust design is required.

#### 4.1.7.1 Naive Overview [MW17]

First, the samples are generated from the Base Sampler, which has a fixed standard deviation (reasonably small) and center. The Base Sampler we will be using is the constant time Knuth Yao Sampler. The generated samples are then combined using the convolution algorithm, which will be described in the later sections. Finally, to achieve any arbitrary center, the binary representation of the center is rounded using the samples from the Gaussian distribution scaled geometrically at every iteration until the sample in an integer is obtained. It is said that the final output is obtained by using the Gaussian distribution samples. The resulting sample also follows Gaussian nature.

The Gaussian Sampler can be divided into two parts the Base Sampler (fixed) and the Rounder, which combines the samples from the fixed sampler and then uses Gaussian rounding to obtain the sample with given standard distribution and center.

#### 4.1.7.2 Rounder

The definition of the Gaussian Sampler remains the same except the addition of a new parameter called the smoothing parameter denoted as $\eta_\epsilon$ (where $\epsilon > 0$ is very small) which is used for tail bound and convolution theorem calculation. The constraints obtained through mathematical proofs are as follows:

Typically $\eta_\epsilon(\mathbb{Z}) > 6$ and $\sigma \geq \eta_\epsilon(\mathbb{Z})$.

Starting with the iterative combination ($SampleI$) of the samples with convolution to obtain a sample with a larger standard deviation. The constraint involved in the

Base Sampler design is the fact that it's standard deviation $\sigma_o \geq \sqrt{2}\eta_\epsilon(\mathbb{Z})$. The algorithm which governs the convolution of the sample adapted from [MW17] is as follows:

**Algorithm 2:** SampleI(i)

**Result:** returns combined sample of a sampler with higher $\sigma$

**if** *i==0* **then**

  $x \leftarrow SampleB_{\sigma_0}(0)$ ;

  return x

**else**

  $x_1 \leftarrow SampleI(i-1)$;

  $x_2 \leftarrow SampleI(i-1)$;

  $zi \leftarrow \lfloor \frac{\sigma_{i-1}}{\sqrt{2}\eta_\epsilon(\mathbb{Z})} \rfloor$;

  ${\sigma_i}^2 \leftarrow ({z_i}^2, max((z_i-1)^2, 1)){\sigma_{i-1}}^2$;

  $y \leftarrow z_i x_1 + max(1, z_i - 1)x_2$;

  return y

**end**

where $SampleB_{\sigma_0}(0)$ is the Base Sampler with standard deviation as $\sigma_0$ and center as 0.

The obtained samples are used by the routine $SampleZ_{b,k,max}$ (where b is the base, k is the binary precision, max is the maximum value of i), which in turn uses the Gaussian center rounding routing $SampleC_b$. Both the routines are adapted from [MW17]. The base b is responsible for the number of a bit being rounded in the k bit binary representation in the center. Furthermore, b is the number of base samplers required initially, which are used during the Gaussian rounding of the center.

**Algorithm 3:** $SampleZ_{b,k,max}(\sigma, \text{c})$

**Result:** returns sample of Sampler with given $\sigma$ and c

**Input**: $\sigma$,c.

$x \leftarrow SampleI(max)$;

$K \leftarrow \sqrt{s^2 - \bar{s}^2}/s_{max}$;

$\tilde{c} = \lfloor c + Kx \rceil_k$;

$y \leftarrow SampleC_b(\tilde{c})$;

return y

where $\bar{s} = s_0(\sum_{i=0}^{k-1} b^{-2i})$, b is the base defined earlier similarly k is the bit precision where the rounding is to be started. Now, k is not the precision involved in the binary representation of the center c. It is said that if the precision is too high, a larger number of samples is required for the Gaussian rounding of the sample, and hence the execution time of the sampler would increase. This execution time is thus reduced using the optimization, which involves rounding the lower significant bit through a simple biased coin flip, and then the remaining k bits of the binary representation is rounded through the Gaussian rounding described below. This biased coin flip rounding is denoted by $\lfloor . \rceil_k$.

**Algorithm 4:** $SampleC_b(c \in b_k\mathbb{Z})$

**Result:** returns the sample after Gaussian rounding the center $\sigma$

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

To understand this algorithm, consider a toy example, as shown below.

$c = 0.0011011b_k \in 2^{-k}\mathbb{Z}$

$x = SampleB_{\sigma_0}(b_k)$

$x = 2x$

$x = (x + b_k)/2^k$

$c = c - x$

$c = 0.0011001 \in 2^{-k+1}\mathbb{Z}$

Here b is taken as 2.

#### 4.1.7.3 Base Sampler

The authors in [MW17] discuss the performance of different Gaussian sampler for the consideration of base sampler. They find Knuth Yao Sampler [DG14] as the

fastest among the samplers. The other reason to prefer the Knuth Yao Sampler is also discussed in the previous sections.

$SampleB_{\sigma_0}(c_l)$ where $c_l = l/2^b$ where $0 \leq l < b$ are the Base Samplers (the Knuth Yao Samplers) with $\sigma$ as the standard deviation and $c_l$ as the $l^{th}$ center and b is the chosen base of the Sampler.

### 4.1.8 Proposed Gaussian Sampler C++ Simulation

Parameters

$\eta_\epsilon(\mathbb{Z}) = 2.5$
$\tau = 5$ *(tailcut)*
$\lambda = 107$
$maxflips = 35$ *(biased coin flip)*
$\sigma_0 = 4.00$ *(Knuth Yao Sampler)*
$Numberofsamplesrecorded = 1000000$
The PRNG source used from NTL C++ library [Sho09].

The simulation was performed for two scenarios.

1. Case I: Fixed center variable Standard Deviation.
2. Case II: Fixed standard deviation variable center.

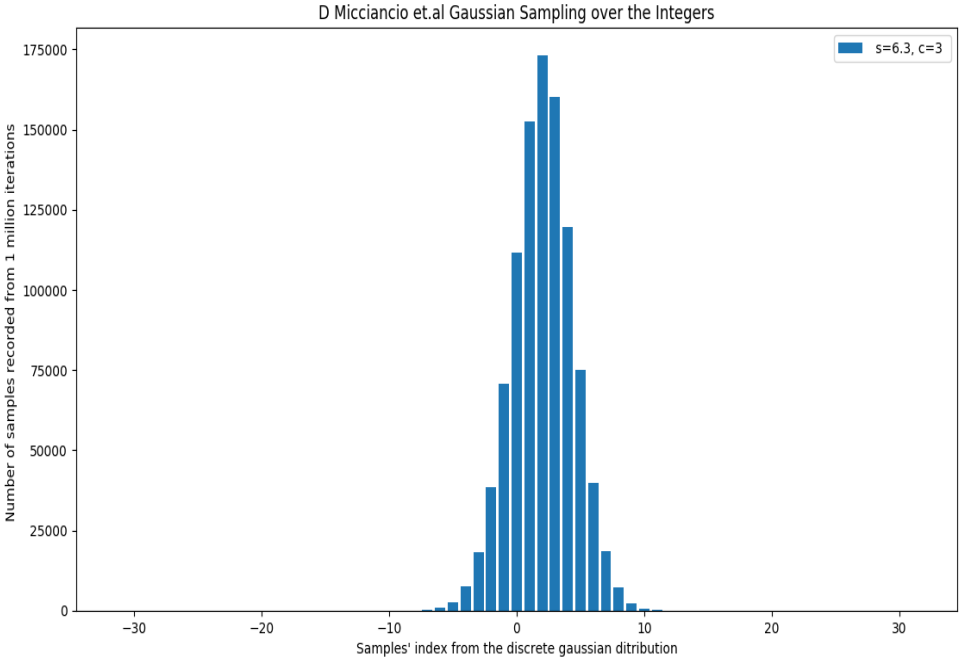


FIGURE 4.9: Case I Histogram for $\sigma = 6.3$ c=3

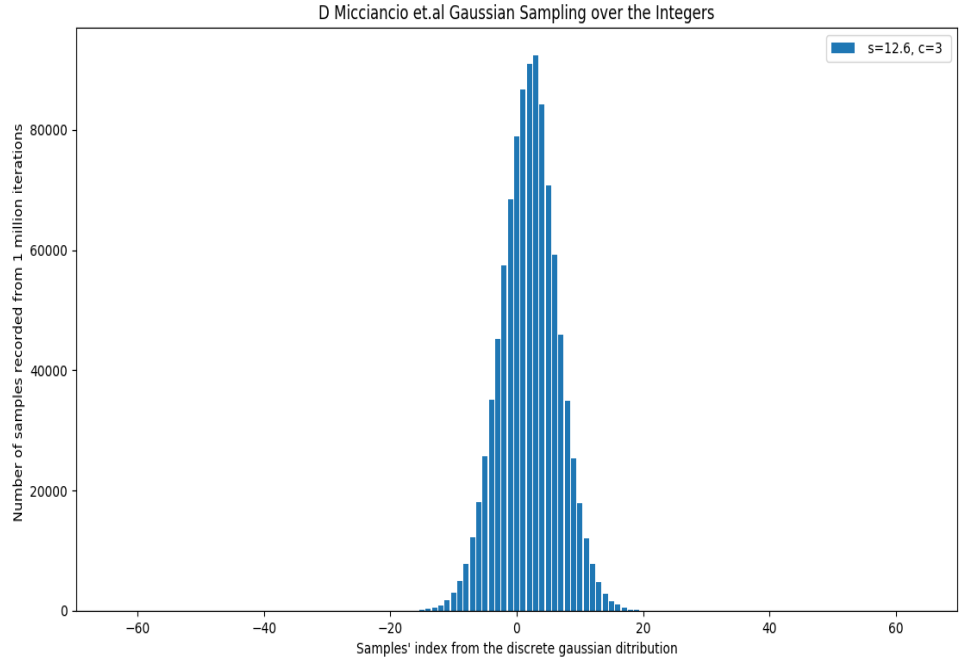


FIGURE 4.10: Case I Histogram for $\sigma = 12.6$ c=3

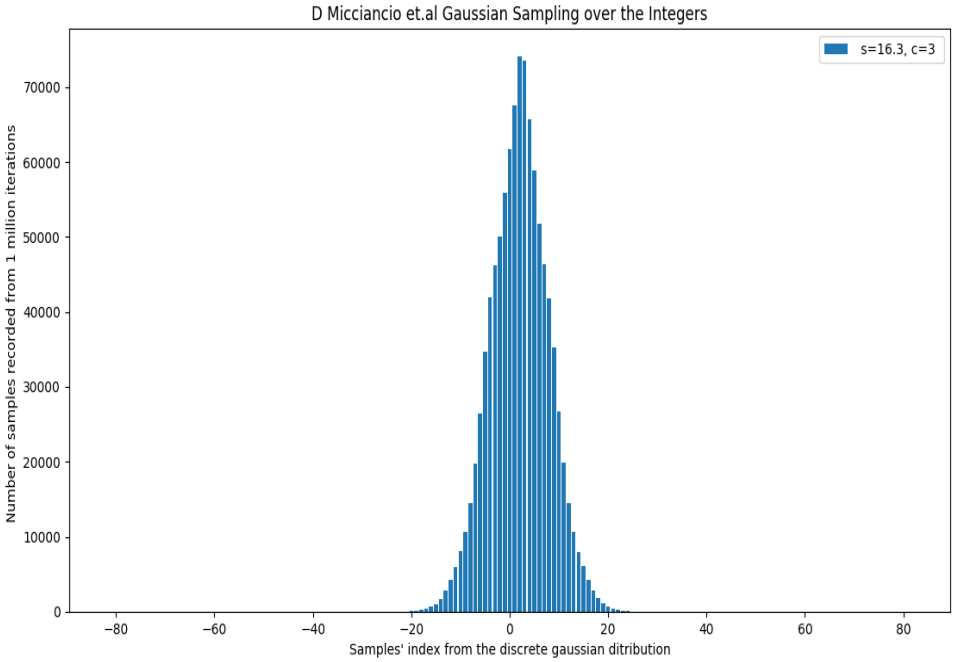


FIGURE 4.11: Case II Histogram for $\sigma = 16.3$ c=3

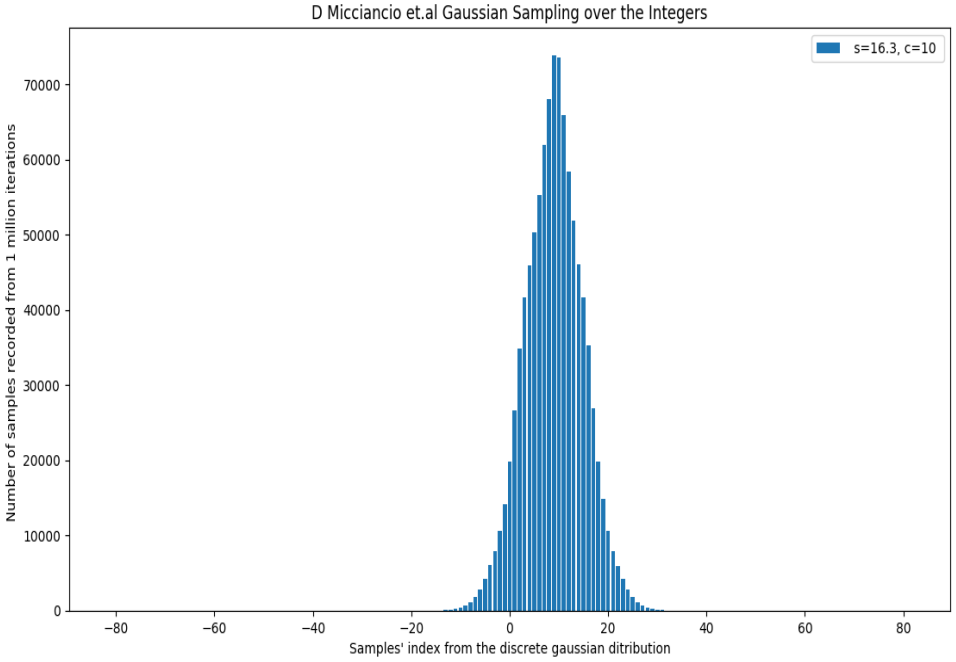


FIGURE 4.12: Case II Histogram for $\sigma = 16.3$ c=10

### 4.1.9 Proposed Gaussian Sampler Hardware Architecture

The hardware architecture for the Gaussian Sampler based on the discussion and the simulated results needs constant time and heavily optimized to minimize resource consumption. The implementation of the Rounder is difficult in the hardware due to its recursive nature when both the standard deviation and center are varying. However, with any of these parameters fixed, there is a scope for hardware implementation of the design. On the other hand, the Base Sampler can be implemented in the hardware as in the literature [KRR+18][Roy17][SRVV14][KRVV19]. The critical point to note is that the number of Base Sampler requirements in the case of the proposed Gaussian Sampler depends on the base b, which governs the performance of the sampler, thus there exists scope for optimization based on the trade-off. Presently implementing such designs, Xilinx Zynq System on Chip (SoC) is a suitable choice because of the hardware-software co-design flexibility provided by the Xilinx FPGA and the ARM Core.

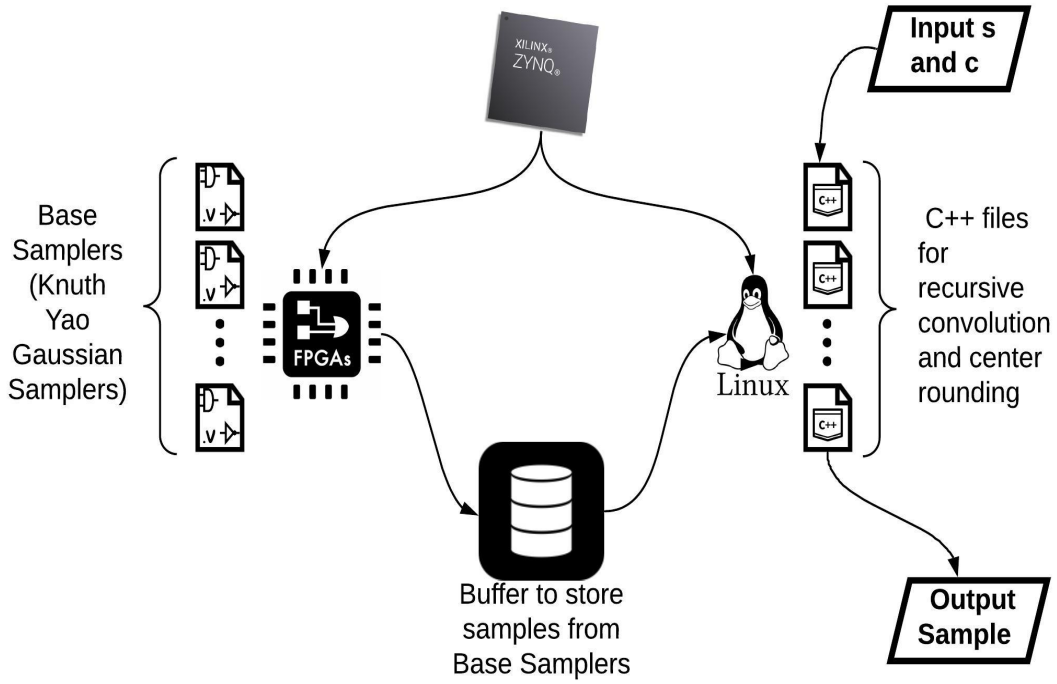


FIGURE 4.13: Hardware Software Co-Design Architecture of Proposed Gaussian Sampler on Zynq SoC.

Since the implementation of the recursive Rounder on the Linux is reasonably trivial with the security concerns and parallelization on the ARM core being the only thing

to be addressed. The multiple Base Sampler implementation on the FPGA opens up the scope for optimization considering the constant time and power leakage resistant design.

#### 4.1.9.1 Base Sampler Optimisation

The constant time Knuth Yao sampler proposed by [KRVV19], as described in the section 4.1.5.3 can be extended along with the usage of multi-level logic optimization [Fuj19] instead of the two-level optimization used by the authors. Tools such as ABC [BM10] or cirkit [SAGD17] can be used for the multi-level logic optimization. The truth table in Figure 4.8 can be modified by inserting another column for encoding the Base Samplers depending on the parameter b. The diagram 4.14 depicts the strategy for optimally implementing the Base Sampler keeping the constraint in mind.

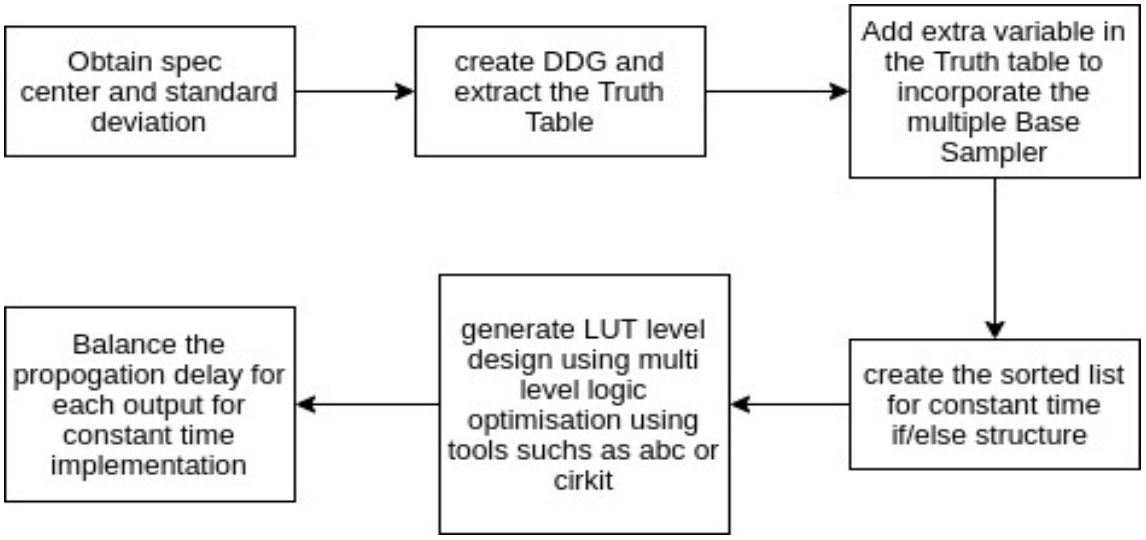


FIGURE 4.14: Base Sampler Implementation flow

# Chapter 5

# Learning with errors (LWE) and Ring LWE (RLWE)

## 5.1 Lattices

Lattices are periodic structure of points in n-dimensional space. The lattices are generated using the n basis which are linearly independent vectors say $b_1, ....., b_n \in \mathbb{R}^n$, here $\mathbb{R}$ is the set of all real values. Mathematically lattices are defined as follows:

$$\mathbb{L}(b_1, ....., b_n) = \{\sum_{i=1}^{n} y_i b_i : y_i \in \mathbb{Z}\} \tag{5.1}$$

The linearly independent vectors used to define the Lattice are called the basis of the lattices. The Lattice based cryptography originates from the hardness of the lattice problems. Some of the hard problems involving lattices are the shortest vector problem (SVP); it's an approximate counterpart approximate-SVP and bounded decision decoding (BDD). It is said that these hard problems involving lattices cannot be solved in polynomial time. The cryptographic constructions through these hard lattice problems are said to have worst-case hardness compared to the factoring based cryptographic constructions, thus providing a strong security guarantee. Presently, there is no known quantum algorithm for solving the lattice problems, thus opening up the era for post-quantum cryptographic constructions.

## 5.2 LWE

The first version of the cryptosystem, along with the underlying security proof, was proposed by Regev [Reg05]. The LWE scheme was further improved by Kawachi et al. [KTX07] and then by Peikert [PVW08]. Through these works, it is evident that with suitable parameters, this problem can be reduced to Lattice-based hard problems, namely the approximate-SVP, which means breaking the LWE cryptosystem implies deducing an effective quantum algorithm to solve the approximate-SVP hard problem. The thesis works by Migliore [Mig17] and Bonnoron[Bon18] provide more theoretical insights on the hardness of the problem. Intuitively the LWE problem can be described as the task to obtain a secret s $\in \mathbb{Z}^n$ from a series of random approximate linear equations on s.

### 5.2.1 LWE definition

The majority of the content has overlap with [Cat18]. Let $n \geq 1$ be the dimension and $q \geq 2$ as the modulus where n,q $\in \mathbb{Z}$. $\xi$ denotes a probability distribution over $\mathbb{Z}_q$ in interval (-q/2,q/2]. $\xi$ is a zero mean Gaussian distribution with standard deviation as $\sigma$ rounded to nearest $\mathbb{Z}$. The parameters involved are [n, q, $\sigma$]. Consider a vector s $\in {\mathbb{Z}_q}^n$. Now, assume a distribution $\mathbb{A}_{s,\xi} \in {\mathbb{Z}_q}^n$ x $\mathbb{Z}_q$ generated by picking a uniformly at random from ${\mathbb{Z}_q}^n$ and e from $\xi$ to create (a, (a,s)+e) $\in {\mathbb{Z}_q}^n$ x $\mathbb{Z}_q$. Now the elements in $\mathbb{A}_{s,\xi}$ are random approximate linear equation. Now, it is said that for correctly chosen parameters [n, q, $\sigma$] for an algorithm to solve LWE means to guess the vector s $\in {\mathbb{Z}_q}^n$ with high probability. Form the present state of the art it is difficult to solve the underlying hard problem as discussed in the previous paragraphs.

### 5.2.2 LWE Cryptosystem Construction

The parameters involved in an LWE instance are [n, q, $\sigma$].

1. GenSecretKey($\lambda$): uniformly sample s $\in {\mathbb{Z}_q}^n$.
   sk=s.

2. GenPublicKey(sk, w): uniformly sample w vectors $a_1, ....., a_w \in \mathbb{Z}_q{}^n$ and m errors form the distribution $\xi$ as $e_1, ....., e_w$.
   pk=$(a_i, b_i)_{i=1}{}^w$, where $b_i = (a_i, s)/q + e_i$.

3. Encrypt(pk, w, m$\in$ {0,1}) : m is the binary message, choose S as subset of w public key elements $(a_i, b_i)_{i=1}{}^w$.
   $c_0 = \sum_s a_i$ and $c_1 = m/2 + \sum_s b_i$.
   ct=$(c_0, c_1) \in \mathbb{Z}_q{}^n$ x $\mathbb{Z}_q$,

4. Decrypt(sk,ct): r=$c_1$-($c_0$,s)/q.
   if(r is closer to 0 than 1/2)
   return m=0
   else
   return m=1

#### 5.2.2.1 Problems with LWE scheme

The particle application of LWE includes handling a large number of variables, which increases memory usage, degrades the performance, and comes with a great cost of communication if remote processing is used. All the operation involves matrix multiplication; thus, the latency increases drastically. The parameters are typically very large in the case of Fully Homomorphic Encryption Scheme, such as [n=320,q=4093, $\sigma$=8.00]. All the reasons mentioned above made the research community focus on algebraic structures like polynomial rings, which helps to leverage the polynomial multiplication using Fast Fourier Transform (FFT) or Number Theoretic Transform (NTT).

## 5.3 RLWE

RLWE is an enhanced version of the LWE problem incorporating algebraic structures through polynomial rings over finite fields. This technique was first formulated by Lyubashevsky et al. [LPR12]. The RLWE uses a special structured type of lattices termed as ideal lattices.

### 5.3.1 RLWE definition

Similar to the LWE problem, the underlying hardness lies with the hard to solve lattice problems. The RLWE can be reduced to these hard problems; discussion on the hardness is out of scope for the thesis. Let $\mathbb{R}$ as a ring of n degree over $\mathbb{Z}$, q the prime $\in \mathbb{Z}^+$ and $\xi$ be a probability distribution over $\mathbb{R}_q = \mathbb{R}/q\mathbb{R}$, which means the quotient ring defined by the modulus q. The readers are expected to have a little background in the finite field and rings. For the sake of simplicity, assume $\mathbb{R}_q$ as the set of polynomial with integer coefficient modulo q. The probability distribution $\xi$ can be assumed as sampling n coefficients from the Gaussian distribution in the interval [-q/2,q/2], with the standard deviation $\sigma$, these n coefficient are used to construct a polynomial of $\mathbb{R}_q$. Now, obtain a polynomial s $\in \mathbb{R}_q$, let $\mathbb{A}_{s,\xi}$ be the probability distribution over $\mathbb{R}_q$ x $\mathbb{R}_q$ which contains the tuple (a, (a.s+e) mod q) $\in$ $\mathbb{R}_q$ x $\mathbb{R}_q$, where a $\in \mathbb{R}_q$ is sampled uniformly at random from $\mathbb{R}_q$ and e $\in \mathbb{R}_q$ obtained according to the distribution $\xi$. Now, the RLWE problem states that an algorithm solves RLWE with parameters as [n, q, $\xi$] if for any s $\in$ and given any number of $\mathbb{A}_{s,\xi}$ samples it outputs s with significant-high probability. This is the search version of the problem very simpler to the LWE problem discussed above.

### 5.3.2 RLWE Crytosystem Construction

A very simple encryption scheme present in [LPR12] through the RLWE problem is discussed in this section.

First of all, choose a ring $\mathbb{R} = \mathbb{Z}_q[\mathbb{X}]/\mathbb{F}(\mathbb{X})$, where $\mathbb{F}(\mathbb{X})$ is an irreducible polynomial of degree n, n is chosen as a power of 2 and q is a prime such that q $\equiv$ mod 2n. The typical choice of $\mathbb{F}(\mathbb{X})$=$x^n$+1. Consider the probability distribution as $\xi$ and choose a plaintext modulus t as a ring $\mathbb{R}_t$, which is used to encode the message before encryption.

- SecretKeyGen($\lambda$): sample s from $\xi$ such that s $\in \mathbb{R}_q$.
  sk=s.
- PublicKeyGen(sk): uniformly sample a at random from $\mathbb{R}_q$, e from $\xi$.
  $p_0$=$[-(a.s + e)]_q$.

$p_1$=a.
pk=$(p_0, p_1) \in \mathbb{R}_q$ x $\mathbb{R}_q$.

- Encrypt(pk, m $\in \mathbb{R}_t$): $\Delta = \lfloor \frac{q}{t} \rfloor$, and u, $e_1$, $e_2$ from $\xi \in \mathbb{R}_q$.
  $c_0 = [p_0 u + e_1 + \Delta m]_q$.
  $c_1 = p_1 u + e_2$.
  ct=$(c_0, c_1) \in \mathbb{R}_q$ x $\mathbb{R}_q$.
- Decrypt(sk,ct): m$= \left[ \lfloor \frac{t}{q} [c_0 + c_1 s]_q \rceil \right]_t$.

The correctness bound is governed by the parameters t, q, and $\Delta$.

### 5.3.3 RLWE Implementation

The degree n and the modulus q can be very large for better security and correctness with n in the order of thousands wit modulus q in the order of hundred. As seen from the cryptosystem discussed earlier, the major operations involved are as follows:

- Polynomial Multiplication
- Polynomial Reductions
- Polynomial Additions
- Scale and Rounds
- Modular Reductions

It can be said the polynomial addition doesn't require any sort of optimization and isn't problematic compared to the other counterparts. It is evident that the most operation hungry task is the multiplication of polynomials due to the quadratic complexity using the naive multiplication algorithms. The two classes of the algorithm involved in the optimization are as follows:

- **Class I**: Karatsuba Multiplier [KO62] with complexity as $O(n^{1.585})$ and Toom Cook algorithm extended to polynomial [Coo66] with $O(n^{1.465})$ as the complexity.

- **Class II**: The Number Theoretic Transform (NTT) [Pol71] and popular Fast Fourier Transform with asymptotic complexity as $O(nlogn)$.

The thesis discusses more on the Class II based optimization involved in the polynomial multiplication through the later sections. It is trivial that polynomial multiplication is followed by the modular reduction of the coefficient to bring back the polynomial in the ring. The typical modular reduction used is the Barret reduction; it will be discussed in the later sections. Finally, the scale and round operation are of linear complexity with n, and usually, it is just a division by q. Before starting the modular arithmetic discussion and the polynomial multiplication, it is important to discuss the Residue Number System (RNS), which is a non-positional representation of using the mutually prime moduli as the basis $q_1, ....., q_k$. Using the RNS, the modular arithmetic modulo $q = \prod_{1 \leq i \leq k} q_i$ is obtained using k smaller and independent modular operations. The reasons for the use of RNS originates from the fact that the large value of q results in some limitation of the classical multi-precision arithmetic. Moreover, the parallelism efficiency provided for additions, subtraction, and multiplication is very beneficial.

#### 5.3.3.1 Modular Arithmetic

The reduction of the polynomial to the ring $\mathbb{R}_q$ after the polynomial multiplication depends a lot on the parameter used, i.e., the choice of q. In the RNS representation, since the $q_i$ are mutually co-primes, if a suitable prime is chosen, the reduction can be performed using modular addition and shifts. Coming to the NTT base polynomial multiplication, which is the state of art polynomial multiplication algorithm, the choice of parameter q results in the type of reduction algorithm used. Typical reduction methods are Barret reduction [Mon85] and Montgomery reduction [Bar86].

### Barret Reduction

It uses the simple idea which converts the c≡a mod b to c=a-b.$\lfloor \frac{a}{b} \rfloor$. The following algorithm shows the Barret Modular Reduction.

**Algorithm 5:** Barret Modular Reduction [DR06]

**input**: a,b $\in \mathbb{Z}$ such that $0 \leq a < b^2$, b $> 1$; $\mu = \lfloor \frac{\beta^{2m}}{b} \rfloor$; m= $\lceil log_\beta b \rceil$

**output**: c $\equiv$ a mod b

$q \leftarrow a.\mu$;

$q \leftarrow \lfloor q/\beta^m \rfloor$;

$q \leftarrow q.b$;

$c \leftarrow a - q$;

**if** $c < 0$ **then**

  $c \leftarrow c + b$;

**else**

**end**

**return c**

### Montgomery Reduction

The constraints involved in Montgomery reduction include b being odd and $0 \leq$ a $< b^2$ when finding reduction a modulo b. The algorithm computes the residue of the input scaled by a constant instead of finding the residue directly. The following

algorithm describes Montgomery Reduction.

**Algorithm 6:** Montgomery Modular Reduction [Mon85]

**input**: a,b $\in \mathbb{Z}$ such that $0 \leq a < b^2$, b $> 1$; $\rho$= -1/$n_0$ mod b

**output**: c $\equiv \beta^{-k}$a mod b

**for** *i=0 up to k* **do**
- $\mu_i \leftarrow a_i.\rho \bmod \beta$;
- $u \leftarrow 0$;
- **for** *j=0 to k* **do**
  - $\tilde{r} \leftarrow \mu.b_j + a_{i+j} + u$;
  - $a_{i+j} \leftarrow \tilde{r} \bmod \beta$;
  - $u \leftarrow \lfloor \frac{\tilde{r}}{\beta} \rfloor$;
  - **while** $u > 0$ **do**
    - $j \leftarrow j + 1$;
    - $a_{i+j} \leftarrow a_{i+j} + u$;
    - $u \leftarrow \lfloor \frac{a_{i+j}}{\beta} \rfloor$;
    - $a_{i+j} \leftarrow a_{i+j} \bmod \beta$;
  - **end**
- **end**
- $c \leftarrow \lfloor \frac{a}{\beta^k} \rfloor$;
- **if** $c \geq b$ **then**
  - $c \leftarrow c - b$;
- **else**
- **end**

**end**

**return c**

#### 5.3.3.2 Polynomial Multiplication FFT and NTT

The majority of Homomorphic Encryption scheme architecture opt for NTT based polynomial multiplication as it is also called FFT over the finite field, which means it does not require floating point roots of unity values. However, NTT requires some precomputed values known as twiddle factors. Roy et.al [RVM+14] and Pöppelmann et.al [PNPM15] modified the typical NTT architecture through certain optimisation. Pöppelmann et al. used cache memory to improve the performance of the NTTs,

and SS Roy et.al. Used RNS for parallelized architecture. Further, Öztürk et.al came up with two versions of RNS based NTT multiplier [ODSS15] [ÖztürkDSS17]. In the first version, the computed twiddle factors were sent to the processor yielding to higher communication cost, whereas in the second version, the computed twiddle factors were stores in the BRAM of the FPGA. Moreover, Cousins et al. [CRS17] used a copy of twiddle factors to come up with a high throughput design. The main issues involved in the implementation of the NTT polynomial multiplier are the increase in the routing coefficient requirement as n increase, the management of twiddle factors, and the problem with scale and round operation with specific values of q allowed by the NTT multiplier. For a better understanding of the state of the art NTT based polynomial multipliers, it is better to build up from the DFT.

**DFT**

The Discrete Fourier Transform (DFT) using the convolution theorem can help to convert convolution to simple point wise product (convolution theorem). Let x be a vector of length n and $\omega$ be a primitive n-th root of unity. So for $1 \leq k < n$, the DFT is defined as:

$$X_i = \sum_{j=0}^{n-1} x_j w^{ji} \tag{5.2}$$

Similarly the inverse DFT can be defined as:

$$x_j = \frac{1}{n} \sum_{i=0}^{n-1} X_i w^{-ij} \tag{5.3}$$

Intuitively the naive computation of the algorithm is possible in $O(n^2)$. But a series of optimized algorithm exists. The use of DFT over a field or ring is termed as NTT.

**FFT**

FFT computes the DFT in $O(nlogn)$; it uses the divide and conquers technique to recursively compute DFT by reducing the DFT into smaller subproblems. The length of the vector in these algorithms is of the form $n = 2^k$, but the FFT can be applied by padding zeros at the end until this form is achieved. Moreover, the reordering technique is used in iterative forms of FFT; this reordering technique

swaps each element with its reverse binary index. The following two algorithm [Mil19] shows the FFT using the Decimation in Time (DIT) and Decimation in Frequency (DIF) methods:

**Algorithm 7:** Cooley-Tukey FFT (DIT) [CP05]

**input**: vector x[0:n-1], n-th root of unity $\omega$

**output**: transformed x[0:n-1]

Reorder_vector(x);

**for** *m=1 up to n by 2m* **do**

  **for** *j=0 up to m* **do**

    $a \leftarrow \omega^{\frac{jn}{2m}}$;

    **for** *i=j up to n by 2m* **do**

      $(x_i, x_{i+m}) \leftarrow (x_i + ax_{i+m}, x_i - ax_{i+m})$;

    **end**

  **end**

**end**

**Algorithm 8:** Gentleman-Sande FFT (DIF) [CP05]

**input**: vector x[0:n-1], n-th root of unity $\omega$

**output**: transformed x[0:n-1]

**for** *m=n/2 down to 1 by m/2* **do**

  **for** *j=0 up to m* **do**

    $a \leftarrow \omega^{\frac{jn}{2m}}$;

    **for** *i=j up to n by 2m* **do**

      $(x_i, x_{i+m}) \leftarrow (x_i + x_{i+m}, a(x_i - x_{i+m}))$;

    **end**

  **end**

**end**

Reorder_vector(x);

The Inverse Fast Fourier transform (IFFT) can be calculated using any of the two algorithms by tweaking the n-th root of unity, changing it to its conjugate, and multiplying each value of the vector in x[0:n-1] by 1/n. The Reorder_vector(x) routine is responsible for changing the index. It is also called the scrambling operation where the index is a bit reversed, say 001 (1) changes to 100 (4).

**FFT based polynomial multiplication**

Let assume two polynomials *f(x)* and *g(x)* with degree less than n and coefficients $f_i, g_i \in \mathbb{Z}_p$. The required result is *h=f.g*. The polynomial multiplication is performed through the negative wrapped convolution with co-efficient modulo a prime p, the prime p is of the form $p \equiv 1$ mod 2n by selecting $\theta$ such that $\theta^2 \equiv \omega$ mod p. The negative wrapped convolution is defined as

$$h_i = \sum_{j=0}^{i} h_j g(i-j) - \sum_{j=i+1}^{n-1} f_j g_{n+i-j} \tag{5.4}$$

The negative wrapped convolution using the FFT framework is calculated as

$$\hat{f} = (f_0, \theta f_1, ....., \theta^{n-1} f_{n-1}) \tag{5.5}$$

$$\hat{g} = (g_0, \theta g_1, ....., \theta^{n-1} g_{n-1}) \tag{5.6}$$

$$h = IFFT^{-1}(FFT(\hat{f}), FFT(\hat{g})) \tag{5.7}$$

Further work on FFT Integer multiplication by Schönhage-Strassen introduces multiplication of large integers through an application of FFT with the complexity of O(nlognloglogn). The FFT modular polynomial multiplication can be applied using this algorithm through the negative wrapped convolution of the decomposed integers.

**Algorithm 9:** Modular Polynomial Multiplication using FFT

**input**:polynomial f(x) and g(x) with $f_i, g_i \in \mathbb{Z}_p$
**output**: h(x)=f(x).g(x) mod $(x^n+1)$
Obtain primitive n-th root of unity $\omega$
Obtain $\theta$ such that $\theta^2 \equiv \omega$ mod p.
**for** *i=0 up to n* **do**
  $(f_i, g_i) \leftarrow (\theta^i f_i, \theta^i g_i)$ mod p ;
**end**
$f \leftarrow FFT(f, \omega)$;
$g \leftarrow FFT(g, \omega)$;
**for** *i=0 up to n* **do**
  $h_i \leftarrow f_i.g_i$ mod p;
**end**
$h \leftarrow IFFT(h, \omega^{-1})$
**for** *i=0 up to n* **do**
  $h_i \leftarrow h_i.\theta^{-1}$ mod p;
**end**

### NTT

The NTT replaces the $\omega$ (the n-th complex root of unity) using a root of unity in a finite ring $\mathbb{Z}_q$.n and q are described as q≡ 1 mod 2n. Lets assume a ring $S_q$=$\mathbb{Z}_q/(x^n-1)$ and two polynomial in the ring such that a,b $\in S_q$. The multiplication using the NTT can be calculated as follows

$$c = NTT^{-1}_{\omega_n}(NTT_{\omega_n}(a) * NTT_{\omega_n}(b)) \tag{5.8}$$

The parameter $\omega$ in the equation is termed as the twiddle factors with * represent co-efficient wise multiplication. Moreover, during the inverse NTT the coefficient are scaled using $n^{-1}$. The effective multiplication of a,b $\in \mathbb{R}_q$ is possible if the evaluation of a nd b are improved in the roots of f ($\mathbb{R}_q$= $\mathbb{Z}_q[x]/ < f >$ with f=$x^n+1$ and n=$2^k$). The roots of f are $\omega_{2n}^{2j+1}$ for j=0,....,n-1. Looking at the twiddle factor the even exponent give the root of $x^n$+1 it can be written as $\omega_{2n}.\omega_n^j$. These evaluations can be calculated effectively using the n-point NTT (instead of the 2n-point NTT) on scaled polynomial $\tilde{a}(x) = a(\omega_{2n}.x)$ and $\tilde{b}(x) = b(\omega_{2n}.x)$. The coefficient multiplication gives c(x)=a(x).b(x) mod f(x) in the roots of f and the classical inverse n-point NTT yields the result as the coefficient of the polynomial $\tilde{c}(x) = c(\omega_{2n}.x)$. To obtain the

coefficient $c_i$ of c(x) compute $c_i = \tilde{c_i}.\omega_{2n}^{-i}$, the scaling with $n^{-1}$ can be combined in this step. The algorithm describes the iterative fashion NTT transform.

**Algorithm 10:** Iterative NTT [RVM+14]

**input**:polynomial f(x) $\in \mathbb{Z}_q$[x] of degree n-1 and n-th root of unity $\omega_n \in \mathbb{Z}_q$
**output**: Tranformed polynomial F(x) $\in \mathbb{Z}_q$[x] =NTT(f)

$F \leftarrow BitReverse(f)$;
**for** *m=2 up to n by m=2m* **do**
    $(\omega_m) \leftarrow (\omega_n^{n/m})$ ;
    $\omega \leftarrow 1$;
    **for** *j=0 up to m/2-1* **do**
        **for** *k=0 up to n-1 by m* **do**
            $t \leftarrow \omega.F[k + j + m/2]$;
            $u \leftarrow F[k + j]$;
            $\leftarrow$F[k+j] $\leftarrow u + t$;
            $F[k + j + m/2] \leftarrow u - t$;
        **end**
        $\omega \leftarrow \omega.\omega_m$;
    **end**
**end**

# Chapter 6

# Conclusion

## 6.1 Conclusion

The number of optimised Homomorphic Encryption scheme is increasing at a faster rate, thus sticking to primitives and key components of a better alternative. As the primitives can be tweaked and different scheme can therefore be generated. The hardware implementation of the primitives comprises of FPGA Fabrics, and this gives the most optimised implementation of any design on FPGAs. The design tool used for the simulations, programming hardware and verification are Vivado 2019.1 and Xilinx ISE. Diligent Basys 3 FPGA boards with Xilinx Artix 7 are used for testing.

The Random Bit Generators, especially the TRNG's are based on the physical noise sources for the FPGA implementation the noise source is mainly the ring oscillator jitter. The implementation of the TRNGs can be further optimised by choosing the locations (co-ordinates) of the LUTs present in the FPGA, and different typologies can be studied affecting the throughput and the randomness of the raw bit. The post-processing circuit is used to remove the bais form the random raw bit generator from the noise source.

The Gaussian Sampler Architecture is implemented using the novel Knuth Yao Algorithm which is heavily optimised by the authors. This opens the room for the Side-Channel vulnerability. Presently, constant time implementation and certain masking schemes are used along with the standard implementation of Gaussian

Sampler. It could be interesting to look for any optimisation which can be exploited either through side channel or fault analysis. The constant time implementation technique of the Knuth Yao algorithm can be transformed to a logic synthesis problem yielding constant time optimised and efficient designs. The security of the HW/SW co-design proposed needs to be taken into consideration, for example the buffer used should be encrypted to prevent adversary from intruding the design.

The parameters of the RLWE cryptosystem are heavily dependent on the type of HE scheme chosen. Moreover, the underlying block say the polynomial multiplier based algorithms like NTT are also parameter specific making optimisation difficult. The RNS based NTT multiplier offers great deal of flexibility in terms of parallelization, it would be interesting to partition the design for different FPGA preferably running on the cloud server.

## 6.2 Future Goals

The upcoming tasks and goals are enumerated as follows:

- Implementation of Proposed Gaussian Sampler in HW/SW codesign architecture stated.
- Gaussian Sampler testing on Zynq followed by test suit based verification.
- Analyse the optimisations in the Gaussian Sampler to check for Side Channel vulnerability.
- Implementation of Modular Arithmetic and Number Theoretic Transforms on FPGA.
- Extend the design for parallel architecture through deployment on cloud based mutiple FPGA.
- Integration of modules to create a fully homomorphic encryption scheme.

# Bibliography


[Bar86] Paul Barrett. Implementing the rivest, shamir and adleman public key encryption algorithm on a standard digital signal processor. In *Advances in Cryptology -CRYPTO' 86*, pages 311–323. 1986.

[BCIV17] Joppe W. Bos, Wouter Castryck, Ilia Iliashenko, and Frederik Vercauteren. Privacy-friendly forecasting for the smart grid using homomorphic encryption and the group method of data handling. In Marc Joye and Abderrahmane Nitaj, editors, *Progress in Cryptology - AFRICACRYPT 2017*, pages 184–201, Cham, 2017. Springer International Publishing.

[Ber16] Federico Bergami. *Lattice-Based Cryptography*. PhD thesis, Universita di Padova and Universite de Bordeaux, 7 2016.

[Bih97] E. Biham. *: A fast new des implementation in software*. In: Biham, 1997.

[BM10] Robert Brayton and Alan Mishchenko. Abc: An academic industrial-strength verification tool. In Tayssir Touili, Byron Cook, and Paul Jackson, editors, *Computer Aided Verification*, pages 24–40, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[Bon18] Guillaume Bonnoron. *A journey towards practical fully homomorphic encryption*. PhD thesis, Universite Bretange Loire Mathstic, 5 2018.

[Bra11] Vaikuntanathan V. Brakerski, Z. Efficient fully homomorphic encryption from (standard) lwe. *FOCS, pp. 97–106*, 2011.

[BUC19] Utsav Banerjee, Tenzin S. Ukyab, and Anantha P. Chandrakasan. Sapphire: A configurable crypto-processor for post-quantum lattice-based protocols. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(4):17–61, Aug. 2019.

[BYV18] M. Grujic N. Mentens B. Yang, V. Rozic and I. Verbauwhede. Estrng: A high-throughput, low-area true random number generator based on edge sampling. *IACR Transactions on Cryptographic Hardware and Embedded Systems, vol. 2018, no. 3, pp. 267–292*, 2018.

[Cat18] Joël Cathebras. *Hardware Acceleration for Homomorphic Encryption.* Hardware Architecture [cs. AR], 2018.

[CFAF13] A. Cherkaoui, V. Fischer, A. Aubert, and L. Fesquet. A self-timed ring based true random number generator. In *2013 IEEE 19th International Symposium on Asynchronous Circuits and Systems*, pages 99–106, May 2013.

[CGGI19] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Tfhe: Fast fully homomorphic encryption over the torus. *Journal of Cryptology*, Apr 2019.

[Coo66] Stephen. A . Cook. *On the Minimum Computation Time of Functions.* PhD thesis, Harvard University, 1966.

[CP05] R. Crandall and C. Pomerance. *Prime Numbers: A Computational Perspective.* Springer-Verlag, 2005.

[CRS17] D. B. Cousins, K. Rohloff, and D. Sumorok. Designing an fpga-accelerated homomorphic encryption co-processor. *IEEE Transactions on Emerging Topics in Computing*, 5(2):193–206, 2017.

[DG14] Nagarjun C. Dwarakanath and Steven D. Galbraith. Sampling from discrete gaussians for lattice-based cryptography on a constrained device. *Applicable Algebra in Engineering, Communication and Computing*, 25(3):159–180, Jun 2014.

[DL13] L. Ducas and T. Lepoint. *Bliss: Bimodal lattice signature schemes.* CRYPTO 2013, 2013.

[DN12] Léo Ducas and Phong Q. Nguyen. Faster gaussian lattice sampling using lazy floating-point arithmetic. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology – ASIACRYPT 2012*, pages 415–432, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[DR06] T. Saint Denis and G. Rose. Bignum math: Implementing cryptographic multiple precision arithmetic. *Syngress Publishing Inc.*, 01, 2006.

[FD03] Viktor Fischer and Miloš Drutarovský. True random number generator embedded in reconfigurable hardware. In Burton S. Kaliski, çetin K. Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002*, pages 415–430, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[Fol15] J. Follath. Gaussian sampling in lattice based cryptography. *Tatra Mountains Mathematical Publications*, 2015.

[Fuj19] Masahiro Fujita. Basic and advanced researches in logic synthesis and their industrial contributions. In *Proceedings of the 2019 International Symposium on Physical Design*, ISPD '19, page 109–116, New York, NY, USA, 2019. Association for Computing Machinery.

[GBHLY16] Leon Groot Bruinderink, Andreas Hülsing, Tanja Lange, and Yuval Yarom. Flush, gauss, and reload – a cache attack on the bliss lattice-based signature scheme. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *Cryptographic Hardware and Embedded Systems – CHES 2016*, pages 323–345, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[Gen09] Craig Gentry. *A FULLY HOMOMORPHIC ENCRYPTION SCHEME*. PhD thesis, STANFORD UNIVERSITY, 11 2009.

[GLN13] Thore Graepel, Kristin Lauter, and Michael Naehrig. Ml confidential: Machine learning on encrypted data. In Taekyoung Kwon, Mun-Kyu Lee, and Daesung Kwon, editors, *Information Security and Cryptology – ICISC 2012*, pages 1–21, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[Gol06] J. D. J. Golic. New methods for digital generation and postprocessing of random data. *IEEE Transactions on Computers*, 55(10):1217–1229, Oct 2006.

[HKR+18] 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*, 67(3):322–334, 2018.

[Jul14] Sundararajan E. Othman Z. Jula, A. Cloud computing service composition: A systematic literature review. expert systems with applications. *41(8), 3809–3824. https://doi.org/10.1016/j.eswa.2013.12.017*, 2014.

[JWBN14] K. Lauter J. W. Bos and M. Naehrig. Private predictive analysis on encrypted medical data. *Journal of Biomedical Informatics*, 2014.

[KGV16] A. Khedr, G. Gulak, and V. Vaikuntanathan. Shield: Scalable homomorphic implementation of encrypted data-classifiers. *IEEE Transactions on Computers*, 65(9):2848–2858, Sep. 2016.

[KO62] A. Karatsuba and Yu. Ofman. Multiplication of many-digital numbers by automatic computers. 145:293–294, 1962.

[KRR+18] A. Karmakar, S. S. Roy, O. Reparaz, F. Vercauteren, and I. Verbauwhede. Constant-time discrete gaussian sampling. *IEEE Transactions on Computers*, 67(11):1561–1571, 2018.

[KRVV19] 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)*, pages 1–6, 2019.

[KTX07] A. Kawachi, K. Tanaka, and K. Xagawa. Multi-bit cryptosystems based on lattice problems. In *Public Key Cryptography – PKC 2007*, volume 4450, page 315–329. 2007.

[KY76] D. Knuth and A. Yao. *Algorithms and Complexity: New Directions and Recent Results*, chapter The complexity of nonuniform random number generation. Academic Press, 1976.

[LN16] Patrick Longa and Michael Naehrig. Speeding up the number theoretic transform for faster ideal lattice-based cryptography speeding up the number theoretic transform for faster ideal lattice-based cryptography. *IACR Cryptology ePrint Archive report 2016/504*, 2016.

[LPR12] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. Cryptology ePrint Archive, Report 2012/230, 2012. https://eprint.iacr.org/2012/230.

[MBO18] J.-F. Couchot M. Bakiri, C. Guyeux and A. K. Oudjida. Survey on hardware implementation of random number generators on fpga: Theory and experimental analyses. *Comput. Sci. Rev., vol. 27, pp. 135–153*, 2018.

[Mig17] Vincent Migliore. *Hardware Cybersecurity and Design of Dedicated Components for the Acceleration of Homomorphic Encryption Schemes*. PhD thesis, Université de Bretagne Sud, September 2017.

[Mil19] Kevin Millar. *Design of a Flexible Schoenhage-Strassen FFT Polynomial Multiplier with High-Level Synthesis*. Thesis, 2019.

[Mon85] Peter L. Montgomery. *Modular Multiplication without Trial Division*. 1985.

[MW17] Daniele Micciancio and Michael Walter. Gaussian sampling over the integers Efficient, generic, constant-time. Cryptology ePrint Archive, Report 2017/259, 2017. https://eprint.iacr.org/2017/259.

[NDR+19] Hamid Nejatollahi, Nikil Dutt, Sandip Ray, Francesco Regazzoni, Indranil Banerjee, and Rosario Cammarota. Post-quantum lattice-based cryptography implementations: A survey. *ACM Comput. Surv.*, 51(6):129:1–129:41, January 2019.

[NPC13] K. Qin N. Peng, G. Luo and A. Chen. Query-biased preview over outsourced and encrypted data. *Scientific World Journal*, 2013.

[ODSS15] Erdinç Öztürk, Yarkın Doröz, Berk Sunar, and Erkay Savaş. Accelerating somewhat homomorphic evaluation using fpgas. Cryptology

ePrint Archive, Report 2015/294, 2015. https://eprint.iacr.org/2015/294.

[ÖztürkDSS17] E. Öztürk, Y. Doröz, E. Savaş, and B. Sunar. A custom accelerator for homomorphic encryption applications. *IEEE Transactions on Computers*, 66(1):3–16, 2017.

[PDG14] Thomas Pöppelmann, Léo Ducas, and Tim Güneysu. Enhanced lattice-based signatures on reconfigurable hardware. In Lejla Batina and Matthew Robshaw, editors, *Cryptographic Hardware and Embedded Systems – CHES 2014*, pages 353–370, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[Pei10] Chris Peikert. An efficient and parallel gaussian sampler for lattices. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, pages 80–97, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[Pes17] Peter Pessl. Analyzing the shuffling side-channel countermeasure for lattice-based signatures. Cryptology ePrint Archive, Report 2017/033, 2017. https://eprint.iacr.org/2017/033.

[PNPM15] Thomas Pöppelmann, Michael Naehrig, Andrew Putnam, and Adrian Macias. Accelerating homomorphic evaluation on reconfigurable hardware. Cryptology ePrint Archive, Report 2015/631, 2015. https://eprint.iacr.org/2015/631.

[Pol71] John M Pollard. *The Fast Fourier Transform in a Finite Field.* Mathematics of computation, 25(114):365374, 1971.

[PVW08] C. Peikert, V. Vaikuntanathan, and B. Waters. A framework for efficient and composable oblivious transfer. In *Advances in Cryptology (CRYPTO), LNCS*. 2008.

[Reg05] O. Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Proc*, pages 84–93. 2005.

[Roy17] Sujoy Sinha Roy. *Public Key Cryptography on Hardware Platforms: Design and Analysis of Elliptic Curve and Lattice-based Cryptoprocessors.* PhD thesis, KU Leuven, 5 2017.

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978.

[RV01] SOTO J. NECHVATAL J. SMID M. BARKER E. LEIGH S. LEVENSON M. VANGEL M. BANKS D. HECKERT A. DRAY J. RUKHIN, A. and S VO. A statistical test suite for random and pseudorandom number generators for cryptographic applications, 2001.

[RVM+14] Sujoy Sinha Roy, Frederik Vercauteren, Nele Mentens, Donald Donglong Chen, and Ingrid Verbauwhede. Compact ring-lwe cryptoprocessor. In Lejla Batina and Matthew Robshaw, editors, *Cryptographic Hardware and Embedded Systems – CHES 2014*, pages 371–391, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[SAGD17] M. Soeken, L. G. Amarù, P. Gaillardon, and G. De Micheli. Exact synthesis of majority-inverter graphs and its applications. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 36(11):1842–1855, 2017.

[Sho09] V. Shoup. *NTL: A library for doing number theory.* http://www.shoup.net/ntl/, August 2009.

[SRVV14] Sujoy Sinha Roy, Frederik Vercauteren, and Ingrid Verbauwhede. High precision discrete gaussian sampling on fpgas. In *Revised Selected Papers on Selected Areas in Cryptography – SAC 2013 - Volume 8282*, pages 383–401, Berlin, Heidelberg, 2014. Springer-Verlag.

[SSRV14] F. Vercauteren S. S. Roy, O. Reparaz and I. Verbauwhede. Compact and side channel resistant discrete gaussian sampling. *Cryptology ePrint Archive, Report 2014/591*, 2014.

[SSRV19] K. Jarvinen F. Vercauteren S. S. Roy, F. Turan and I. Verbauwhede. Fpga-based high-performance parallel architecture for homomorphic computing on encrypted data. *Cryptol. ePrint Arch., Tech. Rep. 2019/160*, 2019.

[VHKK08] Ihor Vasyltsov, Eduard Hambardzumyan, Young-Sik Kim, and Bohdan Karpinskyy. Fast digital trng based on metastable ring oscillator. In Elisabeth Oswald and Pankaj Rohatgi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2008*, pages 164–180, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[Yan18] Bohan Yang. *True Random Number Generators for FPGAs.* PhD thesis, KU Leuven, 9 2018.

[ZBV12] C. Gentry Z. Brakerski and V. Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ITCS, S. Goldwasser, ed., ACM, New York*, 2012.