Primitives of Homomorphic Encryption: A Design Perspective

MTP Supervisors:- Prof. Debdeep Mukhopadhyay and Prof. Indrajit Chakrabarti

Sudarshan Sharma
15EC32005
Senior Undergraduate ,
Dept. of Electronics and Electrical Comm. Eng. ,
IIT Kharagpur, India.
Website-http://sudarshan-sh.com

Outline

- **Homomorphic Encryption Introduction**
- **Random Number Generators**
- **Gaussian Samplers**
- **RLWE**
- **Future Goals**

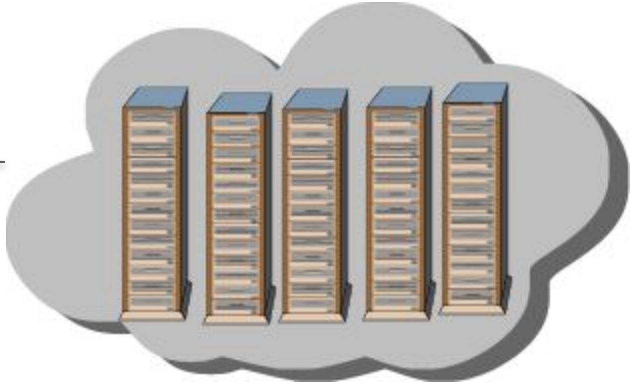
What is Homomorphic Encryption?

"I want to delegate the computation to the cloud, but the cloud shouldn't see my input"

$\text{Enc}_{pk}(x)$

Function f

Enc[$f(x)$]



Client:Alice
(Input :x)

Delegation: Should cost less for Alice to encrypt x and decrypt f(x) than to compute f(x) herself

Server/Cloud
(Function: *f*)

Slide from Shai Halevi: Homomorphic Encryption (Part1):SWHE

Homomorphic Encryption Used Cases

Social Media

Internet Banking Solutions

- Electronics Voting
- Healthcare
- Wearables
    - Smart Watches

Cloud Servers

Motivation

- **Existing Public Key Cryptography Protocols** like RSA and ECC will be insecure by Shor's Algorithm when large scale Quantum Computers are built.

- Need for quantum resistant algorithms
  - Lattice Based Cryptography Suitable Candidate as of now.
    - Why?
      - **Extensive security analysis** as well as **small public key and signature sizes** compared to other post-quantum algorithms

    - Most, lattice-based primitives are based on the hard problem of **finding the solution to linear equations when an error is introduced.**
      - Hard Problem- Learning with Errors (LWE)

Google

nature

Article | Published: 23 October 2019

Quantum supremacy using a programmable superconducting processor

Frank Arute, Kunal Arya, [...] John M. Martinis

Nature **574**, 505–510(2019) | Cite this article

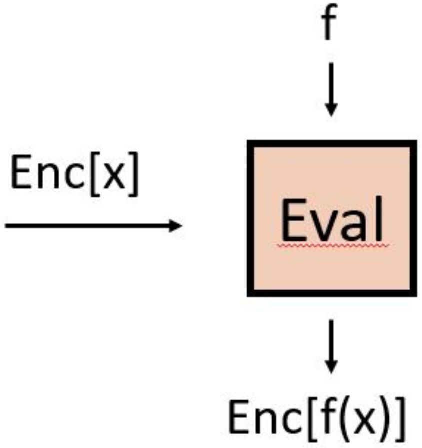
577k Accesses | **5** Citations | **5722** Altmetric | Metrics

Homomorphic Encryption Types

1. Fully Homomorphic Encryption (FHE)
   a. **Arbitrary** Processing
   b. Computationally **Expensive**
   c. Works for **all functions**.

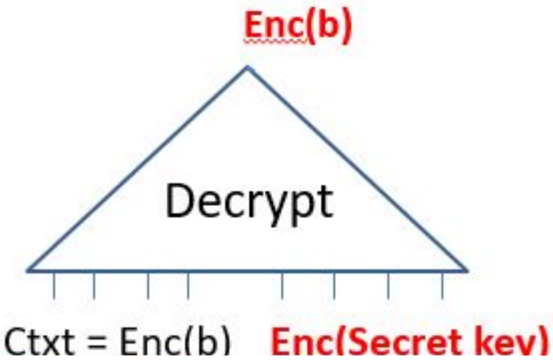
2. Somewhat Homomorphic Encryption (SWHE)
   a. **Limited** Processing
   b. Could be **cheaper** computationally
   c. All pre-2009 schemes were somewhat Homomorphic

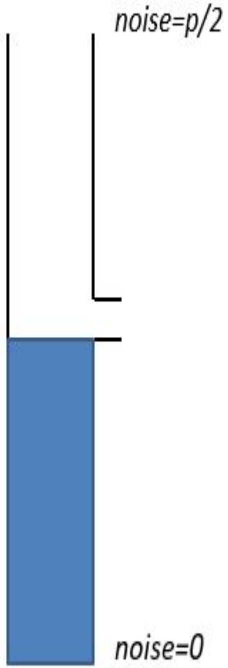
3. Existing constructions of (FHE) schemes start from a (SHE) scheme and use a **complicated mechanism** known as '**bootstrapping**' on top to reduce the noise in the result.



Key Elements of Homomorphic Encryption Scheme

- Learning with Error Scheme(LWE)
  - Preferably Ring Based Learning with Errors (RLWE)
    - Computationally Efficient as Polynomial based compared to matrix based LWE
      - Uses **Ideal Lattices**
- A Gaussian Sampler
- Modular Arithmetic
  - Multiplication of Polynomials
    - **Number Theoretic Transform(NTT)**
  - Division Algorithms for some functions
- Bootstrapping Technique





Slide from Vinod Vaikuntanathan UToronto: Homomorphic Encryption : WHAT, WHY, HOW?

Random Number Generator

Two RNG scheme selected.

- *J. D. Golic.* ***New Methods for Digital Generation and Postprocessing of Random Data****. Computers, IEEE Transactions on, 55(10):1217–1229, 2006.*
    - [Cited in the paper discussed]

- *Yang, B., Rožic, V., Grujic, M., Mentens, N., & Verbauwhede, I. (2018).* ***ES-TRNG: A High-throughput, Low-area True Random Number Generator based on Edge Sampling****. IACR Transactions On Cryptographic Hardware And Embedded Systems*, 267-292.
    - [More Efficient in terms of area and performance]

Golic’s RNG Scheme-I

- **Robust techniques** for generating **high speed** and **high entropy** raw binary sequence by using only logic gates.
- Additional **post-processing of raw binary sequence**

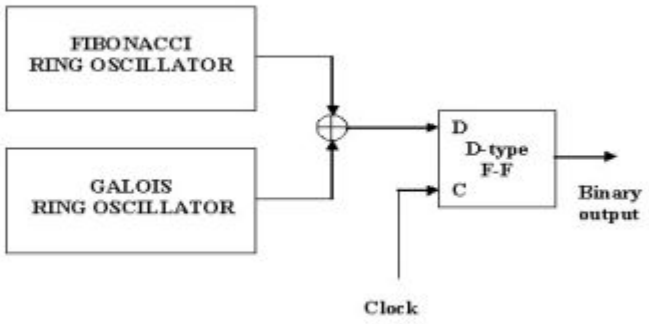


Fig: Digital RNG generator

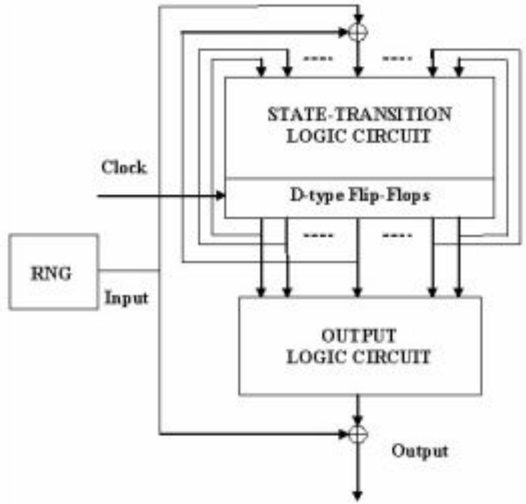


Fig: Generic Post-Processing Circuit

Source:J. D. Golic. New Methods for Digital Generation and Postprocessing of Random Data. Computers, IEEE Transactions on, 55(10):1217–1229, 2006.

Golic's RNG Scheme-II

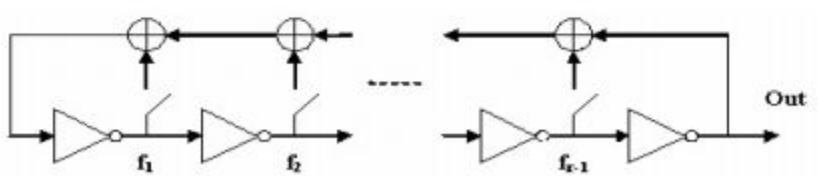


Fig: Fibonacci Ring Oscillator

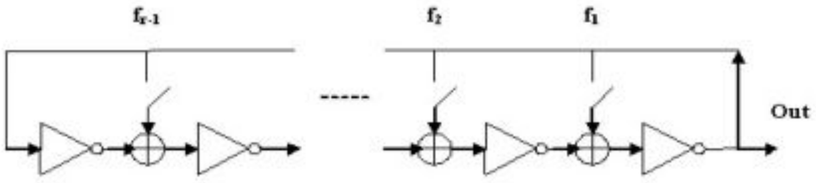


Fig: Galois Ring Oscillator

- The randomness, as well as the robustness, **is increased by XOR-ing the outputs of the two oscillators.**
  - The lengths of the two oscillators minus one should preferably be **mutually prime for randomness.**
  - More randomness **due to metastability may be induced within a sampling unit,** e.g., implemented as a D-type flip-flop.
- **Post Processing through LSFR which is clocked irregularly,** that is if some of its output bits are discarded according to a clock control signal.

Results:-Inferences from Golic's RNG Scheme

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 DRAM	0	0		
LUT as Shift Reg	2	0		
Slice Registers	32	0	41600	0.08
Register as Flip Flop	32	0	41600	0.08

Implementation done on Vivado using **Primitive FPGA fabrics** as the XIlinx synthesis tool at the RTL level was yielding optimised circuit even after using constraints.

Results: Problems associated 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.
    - **Difficult to get the primitive polynomial of higher order**.

- No master clock present in the design
    - Only the long chain of oscillator act as clock
    - After testing the design
        - Conclusions
            - The paper is **quite old** so maybe the LUT-LUT delay would have been greater compared to the present counterparts

ES-TRNG

- Edge Sampling, variable-precision phase encoding and repetitive sampling are used
  - **Edge Sampling**-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**- Only the oscillators phase around the **single edges are sampled** with higher precision.
  - **Repetitive Sampling-Repeats** the sampling at **higher frequency** until the high precision phase region of the oscillator is captured

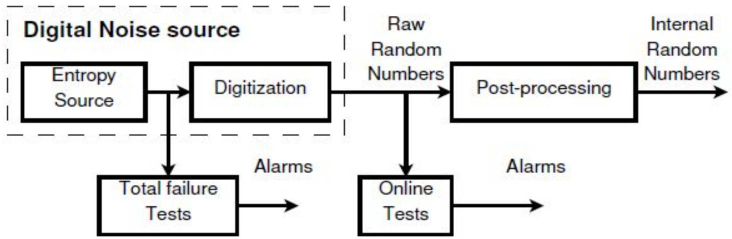


Fig: Generic TRNG Architecture

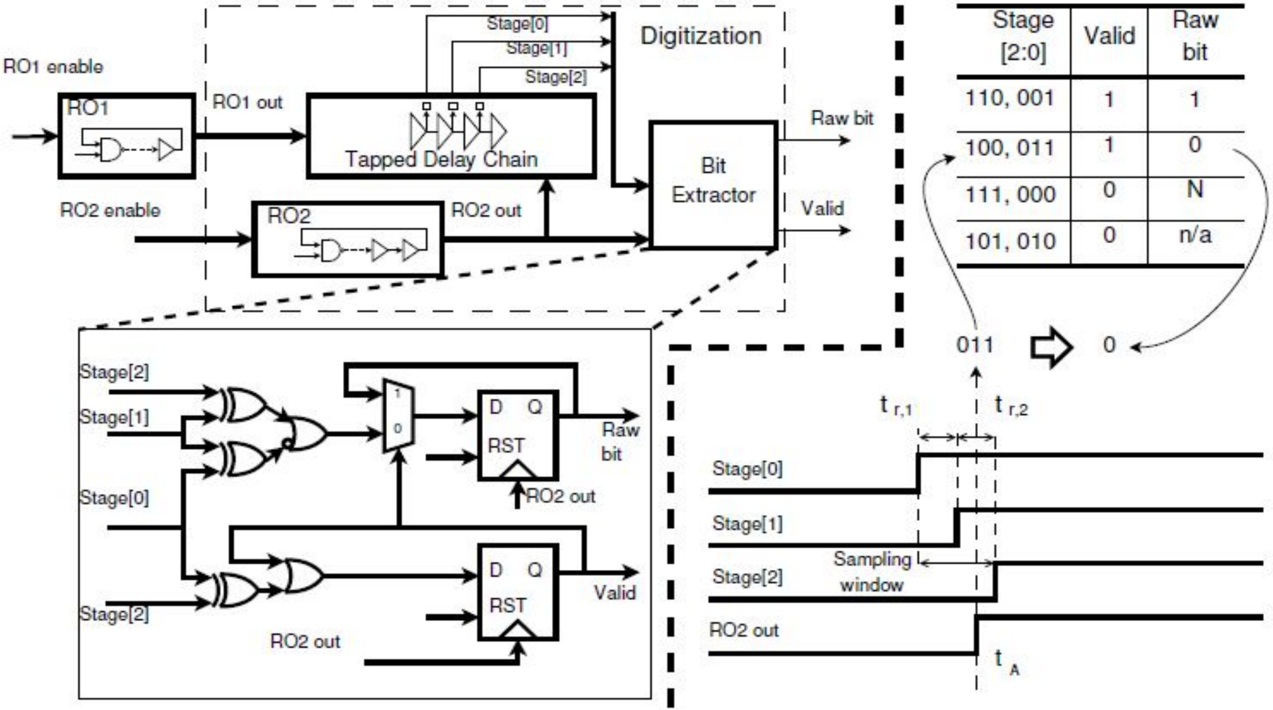
ES-TRNG Important Features

- 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 (lightweight).
    - A compact implementation
    - A reasonably high throughput
    - Feasibility of various implementation platforms
    - Low engineering efforts

- Tested with **AIS-31, a German BSI standard** which put forward a stricter requirement for the design and the evaluation of the TRNGs through a **stochastic model** to evaluate the unpredictability.

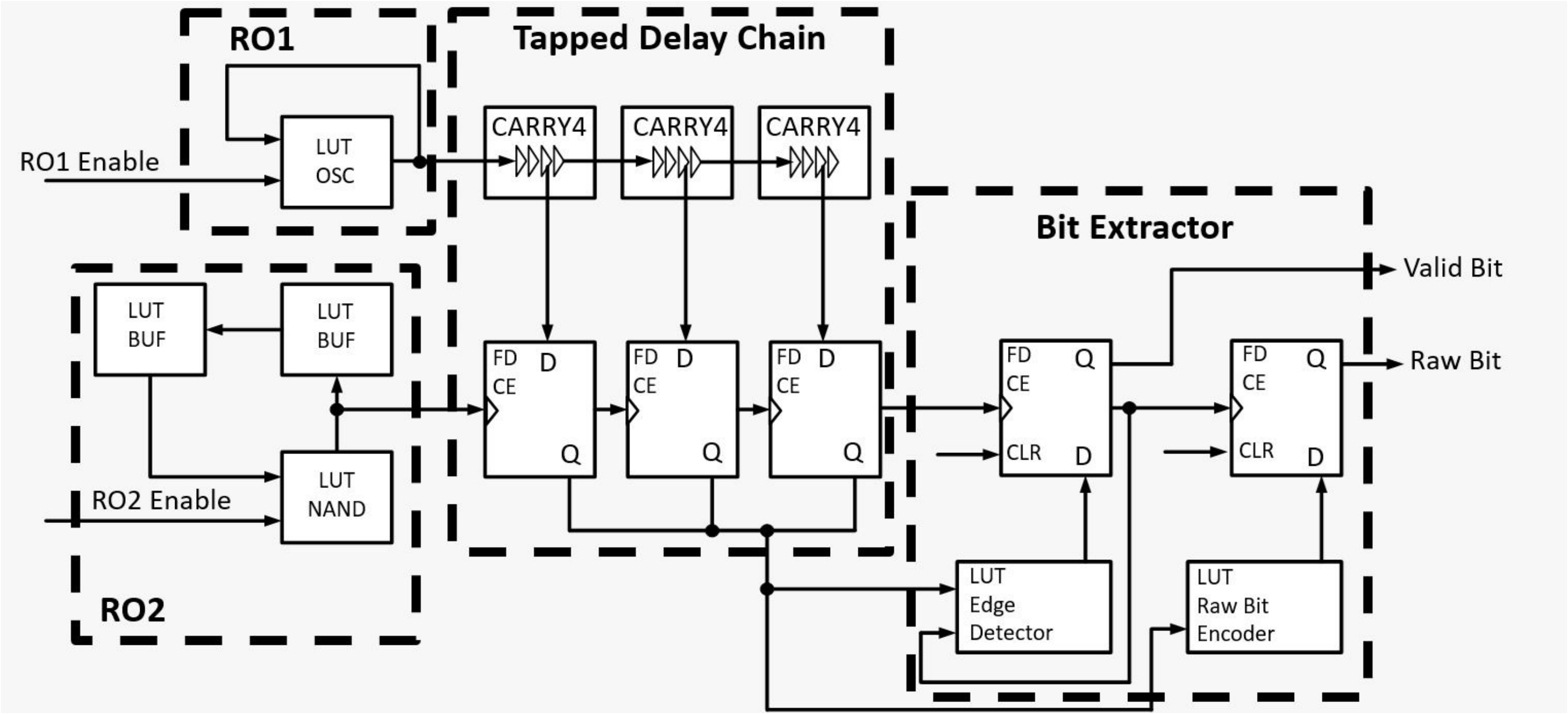
- We know the unpredictability of the TRNG depends on **some physical processes**
    - Here **Timing Phase Jitter** in a free running ring oscillator

ES-TRNG Hardware Architecture



Source*: Yang, B et.al (2018). ES-TRNG: A High-throughput, Low-area True Random Number Generator based on Edge Sampling. IACR TCHES*

ES-TRNG FPGA Architecture



Modified from source: *Yang, B et.al (2018). ES-TRNG: A High-throughput, Low-area True Random Number Generator based on Edge Sampling. IACR TCHES*

Results:-Inferences from ES-TRNG Scheme

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
LUT as DRAM	0	0		
LUT as Shift Reg	0	0		
Slice Registers	5	0	41600	0.01
Register as Flip Flop	5	0	41600	0.01

Implementation done on Vivado using **Primitive FPGA fabrics** as the XIlinx synthesis tool at the RTL level was yielding optimised circuit even after using constraints.

Results:-Evaluation- NIST

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

- NIST SP800-22 as well as the AIS-20/31 tests were performed on approximately 6 million bits.
- Therefore, we could carry out the NIST test on 10 sequences each of size around 600,000 bits.
- NIST Pass Rate: at least 8/10.
- 100 million bit required to pass the failed tests.

Results:-Evaluation- AIS

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

- However, even AIS fails with larger data samples.

Gaussian Sampler

Sinha Roy, S., Vercauteren, F. and Verbauwhede, I.

Sinha Roy, S., Vercauteren, F., & Verbauwhede, I. (2014). High Precision Discrete Gaussian Sampling on FPGAs. *Selected Areas In Cryptography -- SAC 2013*, 383-401. doi:10.1007/978-3-662-43414-7_19

- Introduction and existing sampling methods
- Knuth Yao Algorithm Features
- Contribution of the Paper
- Mathematical Background
- Discrete Distribution Generating Graph
- Efficient Implementation of Knuth Yao Algorithm
- Hardware Architecture
- RNGs

Introduction and Existing Sampling Methods

- **Challenges in Implementation of Discrete Gaussian Distribution**
  - Large number of random bits requirement
  - Lesser statistical distance requirement
    - This requires high precision floating arithmetic or large precomputed tables.

Popular Methods

Rejection

- We generate **points randomly** and then **accept those points that are inside the distribution** and then plot a histogram to obtain the value.
- Slow for gaussian (High rejection rate)
- **Many random bits required** due to high rejection rate for sampled values near the tail

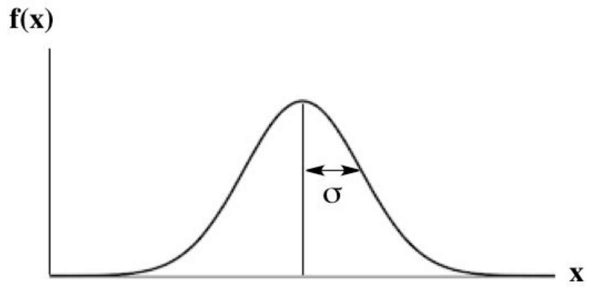
Inversion

- First **generates a random probability(uniform)** and then selects a sample value such that the cumulative distribution up to that sample point is just larger than the randomly generated probability.
- High Precision of random bits.
- Size of Comparator Circuit increases.

Mathematical Background

The continuous Gaussian distribution with variance **σ > 0** and mean **c ∈ R,** with **E ∈ R** as the random variable as x is given as:

$$Pr(E = x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-c)^2/2\sigma^2}$$



For the discrete case over z, with 0 mean and **σ > 0** is given as:

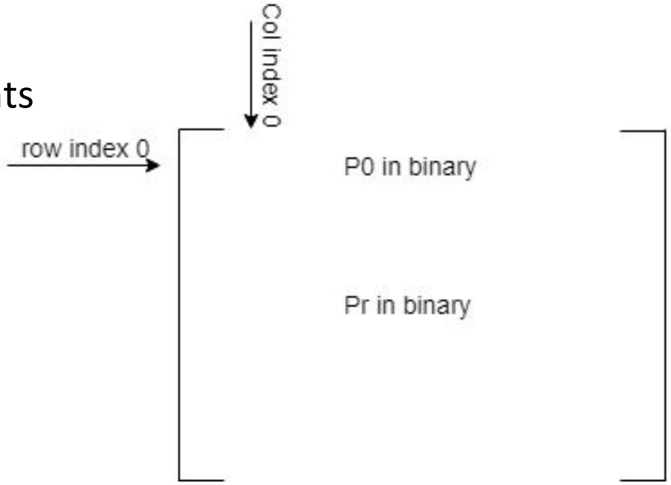
$$Pr(E = z) = \frac{1}{S} e^{-z^2/2\sigma^2} \quad \text{where } S = 1 + 2\sum_{z=1}^{\infty} e^{-z^2/2\sigma^2}$$

- Where S is a **normalisation factor** approximately equal to $\boldsymbol{\sigma}(2\pi)^{0.5}$
- It is calculated here by summing the entire probability values to 1.

Sampling Method & Knuth Yao Algorithm-I

- It uses **random walk on Discrete Distribution Generating (DDG) graphs** to sample non-uniform distribution.

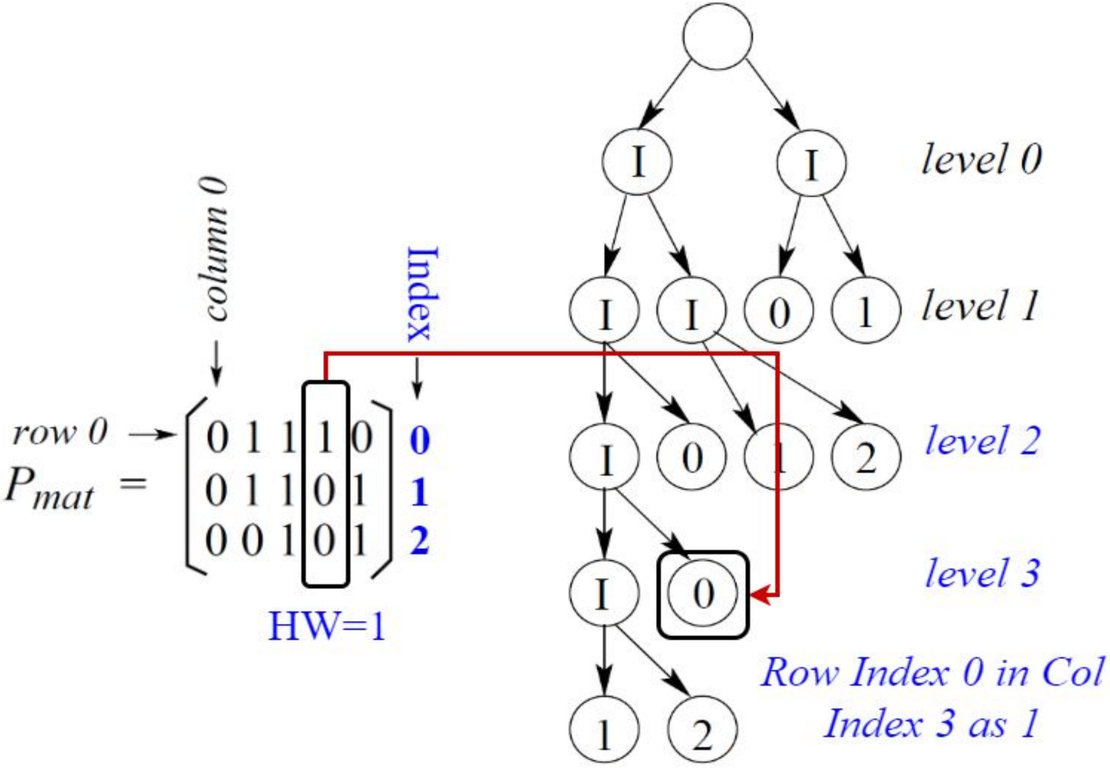
- Let Sample Space for a random variable X consists of n elements
  - 0< **r** <n-1
  - Let p_r be the probability value with respect to **r**
  - We can create a Pmat as follows



- Using this Pmat we can create a DDG.
  - DDG is a **rooted binary tree**
  - It has two types of nodes
    - **Terminal Nodes**
    - **Intermediate Nodes**

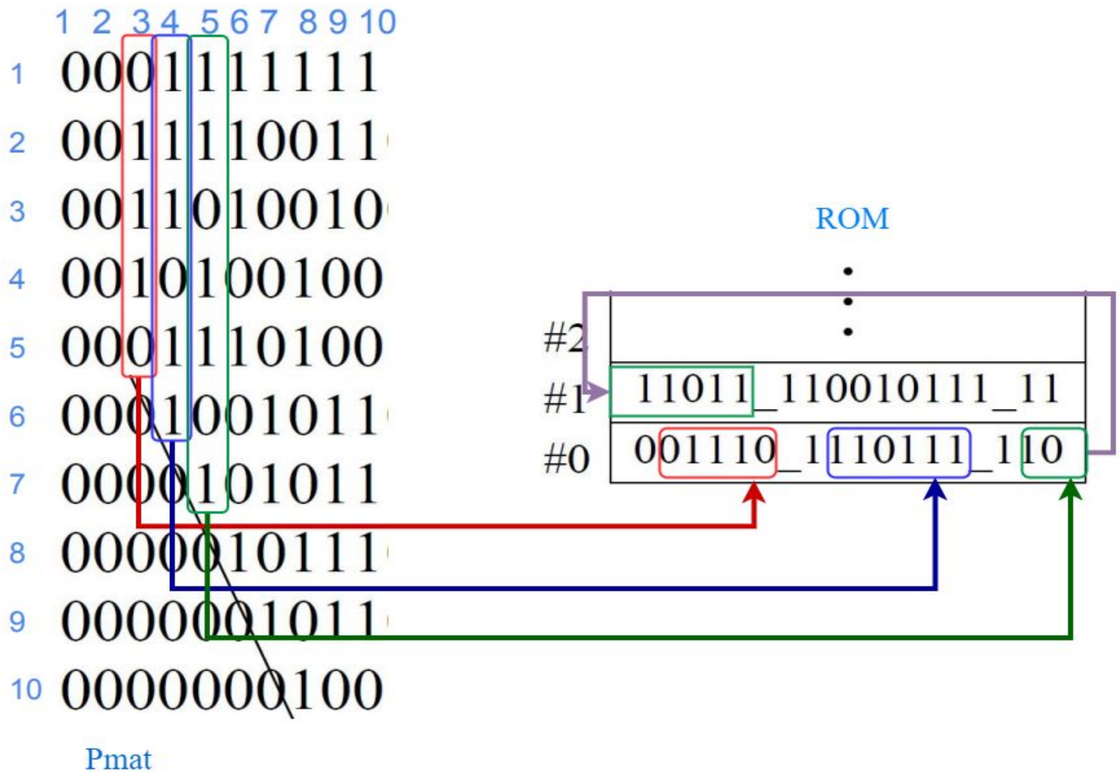
- **Property**:- Number of terminal nodes in a DDG at a level say **i,** is equal to the Hamming Weight of the **i**th column in the probability matrix

Creating DDG- An Example



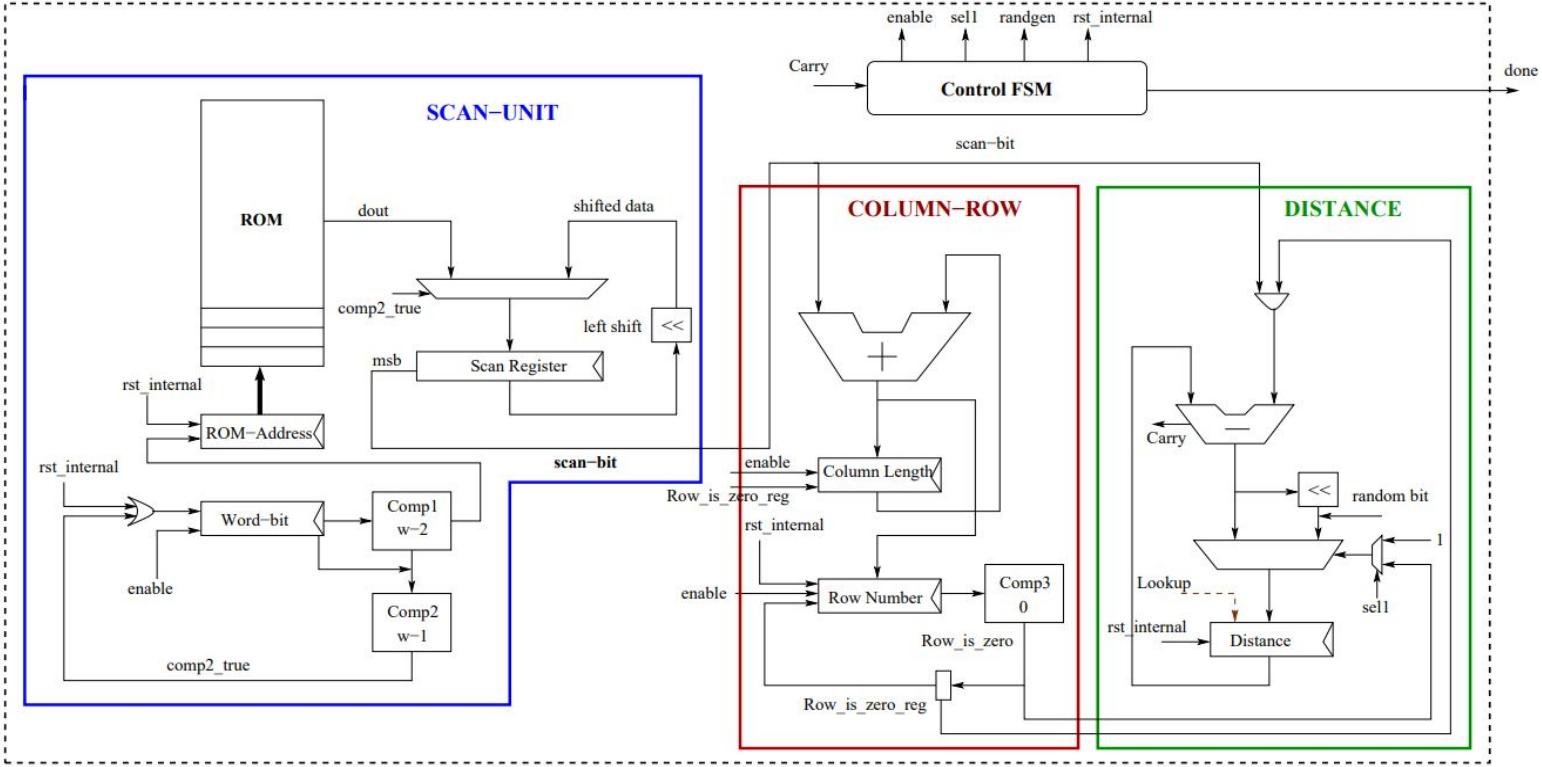
Modified from Source: *Sinha Roy, S., Vercauteren, F., & Verbauwhede, I. (2014). High Precision Discrete Gaussian Sampling on FPGAs. Selected Areas In Cryptography -- SAC 2013, 383-401.*

Efficient Implementation of Knuth Yao Algorithm-III



Modified from Source: *Sinha Roy, S., Vercauteren, F., & Verbauwhede, I. (2014). High Precision Discrete Gaussian Sampling on FPGAs. Selected Areas In Cryptography -- SAC 2013, 383-401.*

Hardware Architecture-III

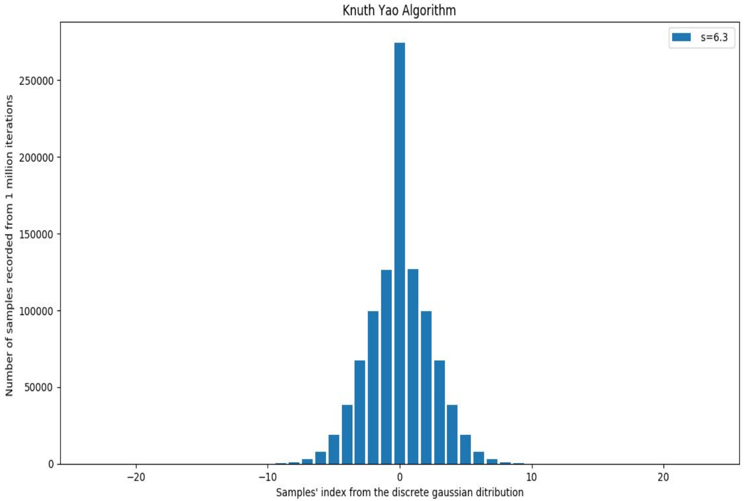


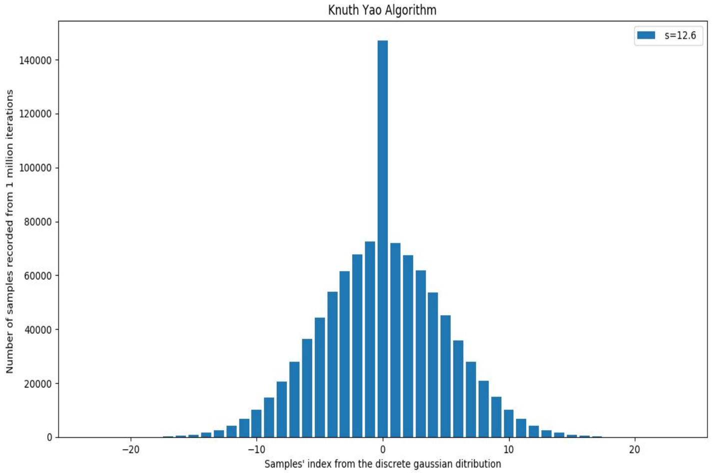
Modified from Source: *Sinha Roy, S., Vercauteren, F., & Verbauwhede, I. (2014). High Precision Discrete Gaussian Sampling on FPGAs. Selected Areas In Cryptography -- SAC 2013, 383-401.*

Knuth Yao Sampling- Results

s=6.3 c=0

s=12.6 c=0





Note: Number of samples recorded from 1 million iterations. [C++ Implementation]
Randomness source:std::random_device.

Constant Time Knuth Yao Algorithm

- S.S. Roy, A.Karmakar et.al proposed two methods for constant time implementation and later optimised it in their successive work. Following are the three papers:
    - *Sujoy Sinha Roy, Oscar Reparaz, Frederik Vercauteren, and Ingrid Verbauwhede.* ***Compact and side channel resistant discrete gaussian sampling****. Cryptology ePrint Archive, Report 2014/591, 2014. https://eprint. iacr.org/2014/591.pdf.*
    - *A. Karmakar, S. S. Roy, O. Reparaz, F. Vercauteren and I. Verbauwhede, "****Constant-Time Discrete Gaussian Sampling****," in IEEE Transactions on Computers, vol. 67, no. 11, pp. 1561-1571, 1 Nov. 2018, doi: 10.1109/TC.2018.2814587.*
    - *Karmakar, Angshuman and Roy, Sujoy Sinha and Vercauteren, Frederik and Verbauwhede, Ingrid, "****Pushing the Speed Limit of Constant-time Discrete Gaussian Sampling. A Case Study on the Falcon Signature Scheme****," in DAC '19, Proceedings of the 56th Annual Design Automation Conference 2019, pp.88:1--88:6, doi:10.1145/3316781.3317887.*

Compact and Side Channel Resistant Discrete Gaussian Sampling: I

- This method caches the first k columns of the probability matrix in a table with 2^k entries. Entries are either:
  - A sample value.
  - An intermediate position in the DDG tree.
- Sampling in two parts, described as following:
  - **Secure Part**: Generates a k bit random index followed by table lookup. Return if sample entry.
  - **Non Secure part**: If table entry is a position in the DDG tree, then a random walk initiates to find a sample.
  - The algorithm **leaks** the absolute values of the samples due to the difference in timing to find a sample.

Compact and Side Channel Resistant Discrete Gaussian Sampling: II

- **Shuffling Technique**: To prevent the leakage a second countermeasure proposed:
  - A random permutation of the leaked and non-leaked samples after the sampling to obfuscate the locations of the samples from the attacker.
- **Bottlenecks**
  - The memory requirement increases exponentially with an increase in the levels of security.
  - Increase in the implementation area.

Constant Time Discrete Gaussian Sampling: I

- Observe a unique mapping between the output sample values and the input random bits of the sampling algorithm.

$$
\begin{aligned}
s_0 &= f^0(r_0, r_1, \cdots, r_{n-1}) \\
s_1 &= f^1(r_0, r_1, \cdots, r_{n-1}) \\
&\vdots \\
s_{m-1} &= f^{m-1}(r_0, r_1, \cdots, r_{n-1})
\end{aligned}
$$

r_0	r_1	r_2	r_3	s_1	s_0	r_0	r_1	r_2	r_3	s_1	s_0
0	0	0	0	0	1	1	0	0	0	1	1
0	0	0	1	0	1	1	0	0	1	1	1
0	0	1	0	0	1	1	0	1	0	1	0
0	0	1	1	0	1	1	0	1	1	1	0
0	1	0	0	0	0	1	1	0	0	0	0
0	1	0	1	0	0	1	1	0	1	0	0
0	1	1	0	0	0	1	1	1	0	1	0
0	1	1	1	0	0	1	1	1	1	0	1

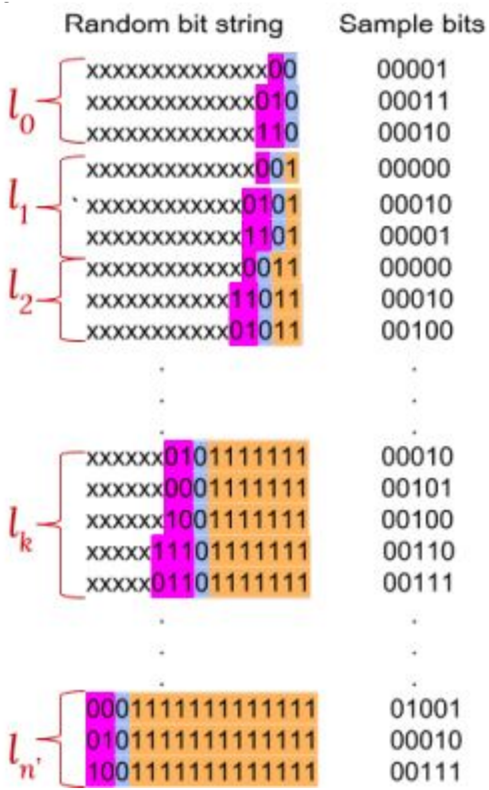
- Express the output sample values as a Boolean function of the input random bits. The samples are encoded in binary. So, each function f above gives either 0 or 1.

Images from source:A. Karmakar, S. S. Roy, O. Reparaz, F. Vercauteren and I. Verbauwhede, "**Constant-Time Discrete Gaussian Sampling**," in *IEEE Transactions on Computers*, vol. 67, no. 11, pp. 1561-1571, 1 Nov. 2018, doi: 10.1109/TC.2018.2814587.

Constant Time Discrete Gaussian Sampling: II

- During sampling, each of these Boolean functions are evaluated in **constant-time**.
- Samples in batches. This increases the throughput.
  - Using Bit-slicing
  - Efficient storage of random bit in registers.
- **Bottlenecks**
  - This method requires a larger program memory to store the formulae f .
  - Since DDG is not uniform some of the random bits are wasted i.e we reach the terminal node without utilising every random bit.

- Extracted a property from the fact that not all random bits(x) it required to attain the sample.



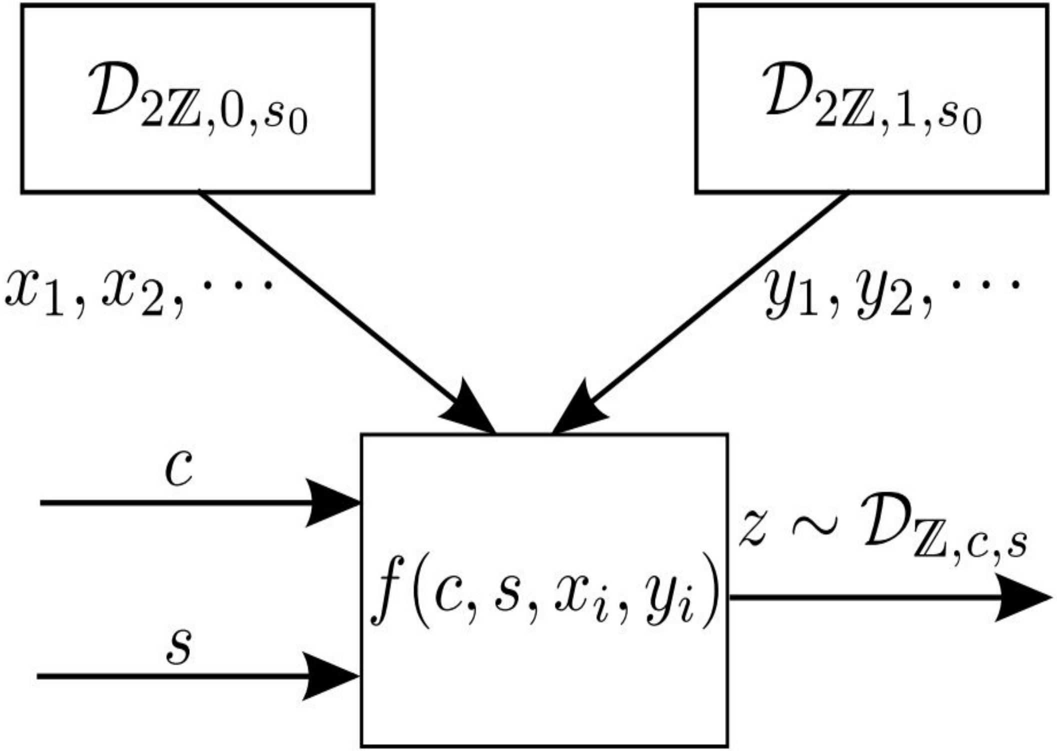
- These lists are created using the following property and then sorted based on value of k.

Theorem 1 *All the random bit strings which generate samples are of the form* $x^i(0/1)^j01^k$, *where* $i, j, k \in \mathbb{Z}^*$, $i+j+k+1 = n$ *and* x *is the* don't care *bits.*

- For each list the boolean functions are created.
  - Constant time execution of if/else used to club the functions.
- Bottlenecks
  - Large Program memory to store boolean functions
  - Lesser modularity.

Gaussian Sampling over the Integers: Efficient, Generic, Constant-Time: I

- Sample with any standard deviation and center.



- Uses Base Samplers which are Gaussian Samplers with small standard deviations.
  - Knuth Yao as base sampler gives best results.

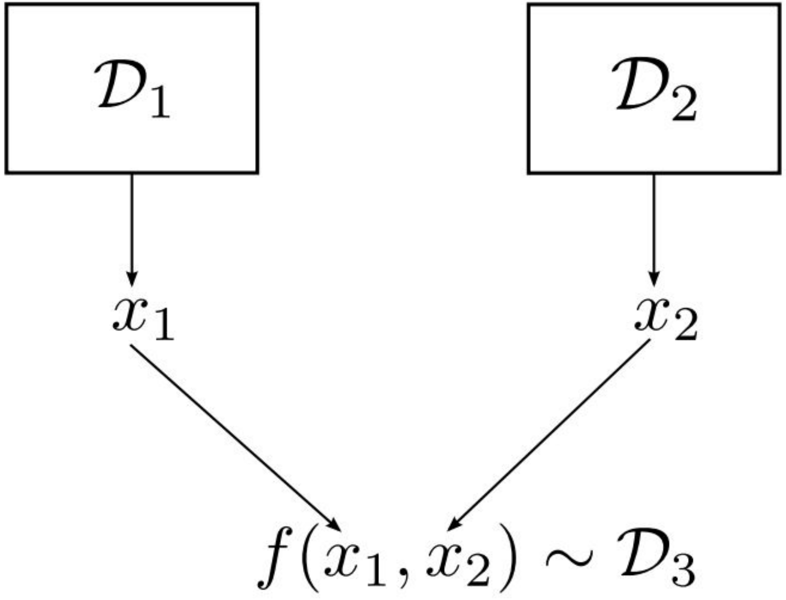
- Scope for constant time as each sample takes same number of iterations.

Image Source: Slides from Michael Walter Presentation at Crypto 2017.

Gaussian Sampling over the Integers: Efficient, Generic, Constant-Time: II

- Two main techniques:

Recursive Convolution



Gaussian Rounding of Center

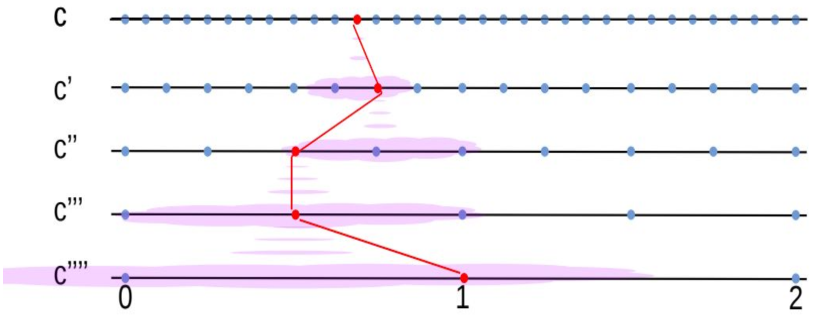


Image Source: Slides from Michael Walter Presentation at Crypto 2017.

Gaussian Sampling over the Integers: Efficient, Generic, Constant-Time: III

- Rounding center using discrete gaussian sampling.

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

$$x \leftarrow \mathcal{D}_{2\mathbb{Z}, b_k, s}$$

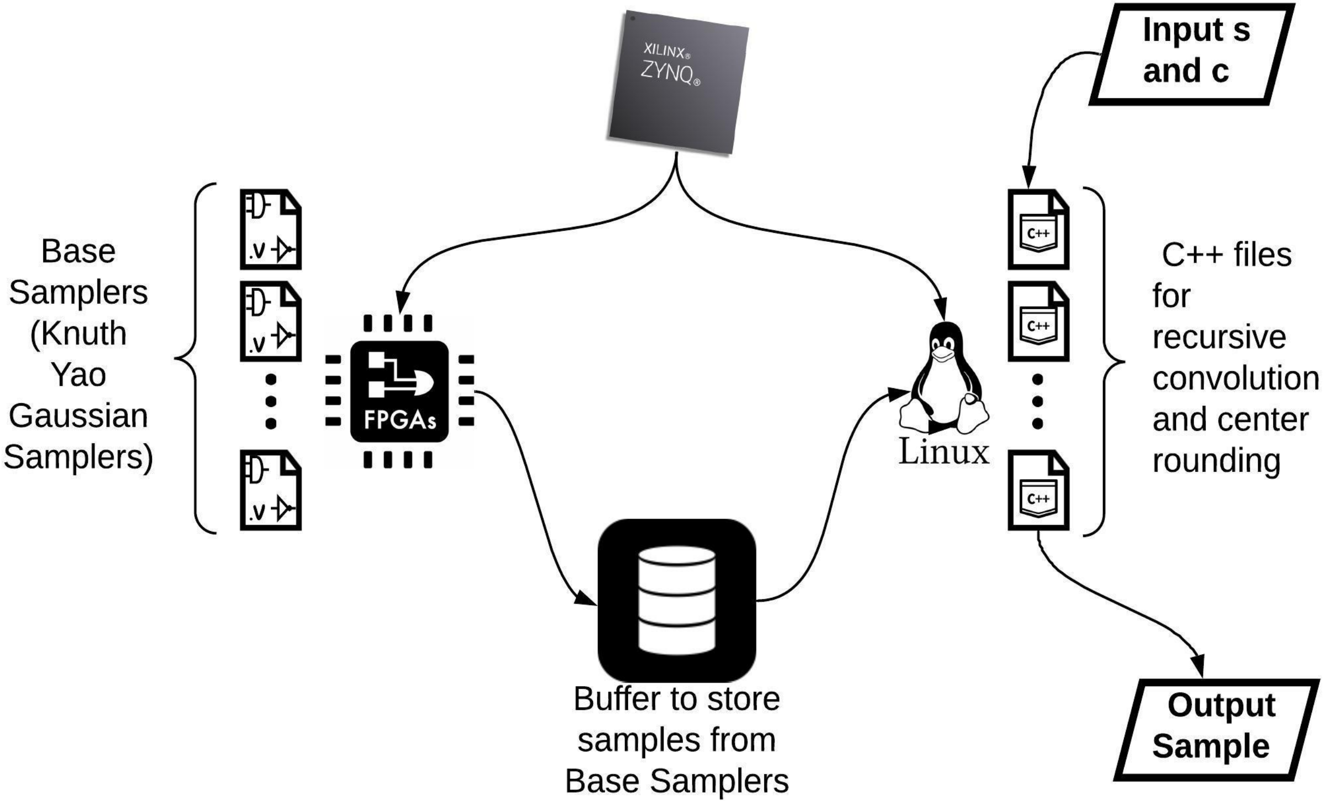
$$x \leftarrow (x + b_k)/2^k$$

$$c \leftarrow c - x$$

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

Image Source: Slides from Michael Walter Presentation at Crypto 2017.

Gaussian Sampling over the Integers: Efficient, Generic, Constant-Time: IV

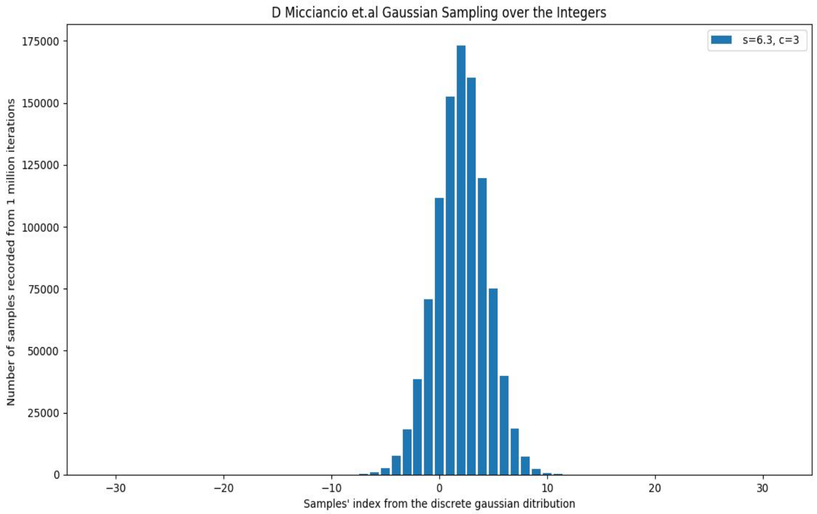


- Proposed Architecture of the Gaussian Sampler.
  - **s** is standard deviation.
  - **c** is the center.

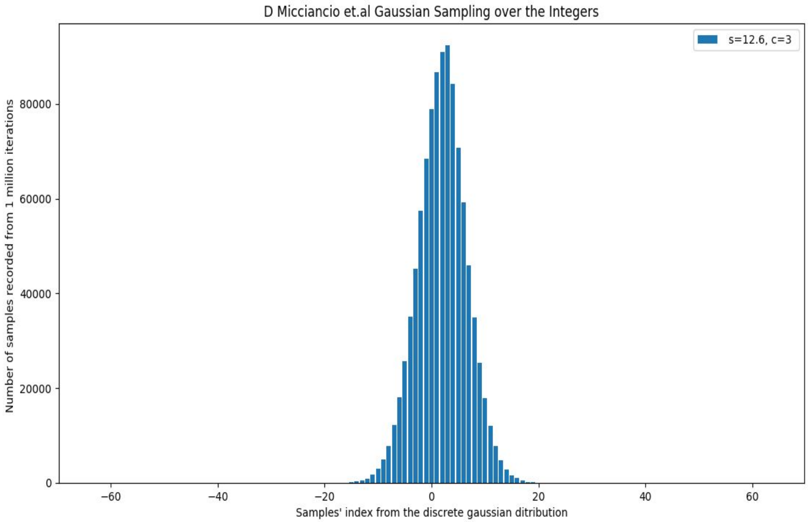
Gaussian Sampling over the Integers: Efficient, Generic, Constant-Time: Results I

Fixed c, s varied.

s=6.3 c=3.0



s=12.6 c=3.0

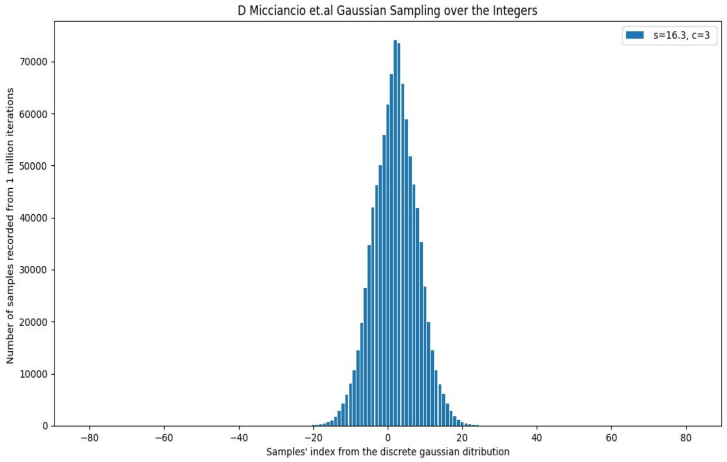


Note: 2 Knuth Yao Base Sampler used. s of Base Sampler 4.01.
Randomness source:std::random_device.
Number of samples recorded from 1 million iterations. [C++ Implementation]

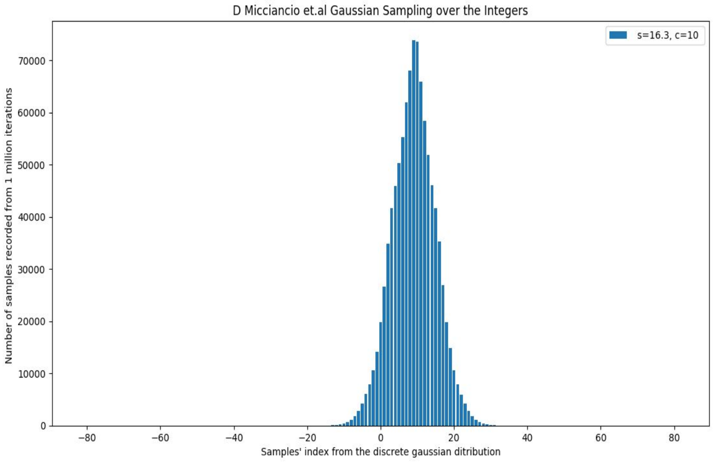
Gaussian Sampling over the Integers: Efficient, Generic, Constant-Time: Results II

Fixed s, c varied.

s=16.3 c=3.0



s=16.3 c=10.0



Note: 2 Knuth Yao Base Sampler used. s of Base Sampler 4.01.
Randomness source:std::random_device.
Number of samples recorded from 1 million iterations. [C++ Implementation]

Ring Learning with Error (RLWE)

- The RLWE scheme is better than the older LWE as:
  - LWE is based on matrix operation
    - Inefficient.
    - Larger key size required.

- RLWE is the algebraic version of LWE.

Definition:

Consider a ring $R_q = Z_q[x]/f$ with $f(x) = x^n + 1$, where n is a power of 2 and the prime q is taken as $q \equiv 1 \bmod 2^n$. The ring-LWE distribution on $R_q \times R_q$ consists of tuples (a, t) with $a \in R_q$ chosen uniformly random and $t = as + e \in R_q$, where $s \in R_q$ is a fixed secret element and e has small coefficients sampled from the discrete Gaussian X_σ.

Ring Learning with Error (RLWE)

- The distribution X_σ may be seen as sampling n coefficients from a normal distribution over $[-q/2, q/2[$, with standard deviation σ, to construct a polynomial of R_q.

- The key elements required for RLWE schemes:

    - Discrete Gaussian sampling for the generation of the error polynomials.

    - Polynomial arithmetic units.
        - Addition/Subtraction.
        - Multiplication
            - NTT better than FFT as only integers are used.[**Costly**]
            - Reduction Modulo f(x).

    - Division-and-round unit for computing homomorphic multiplications

Future Plan

- Planning to employ a parallelised implementation of the NTT-based polynomial multiplication in multiple FPGA (preferably in FPGA-cloud setting).
- Working on to incorporate the (Residue Number System) RNS with NTT for faster arithmetic processing.
- Get the Hardware/Software architecture of Gaussian Sampler proposed working.
- Extend the RLWE design for Encryption/Decryption Scheme of Fully Homomorphic Encryption.
- Modified compact design of ES-TRNG through manual LUT placement.

End of Presentation

Thank You!