Primitives of Homomorphic Encryption
A Design Perspective

Sudarshan Sharma
15EC32005
Senior Undergraduate ,
Dept. of Electronics and Electrical Comm. Eng. ,
IIT Kharagpur, India.
Website-http://sudarshan-sh.com
Email-sudarshansharma04@gmail.com

Outline

- Motivation
- What is Homomorphic Encryption(HE)?-Very Brief
  - Types
  - Elements of HE
  - Gaussian Sampler
    - Knuth Yao Algorithm
      - Efficient Implementation
      - Hardware Architecture
    - Random Number Generators
      - Golic's RNG Scheme
      - ES-TRNG
- Present Design Status
- Conclusion

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
    - 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

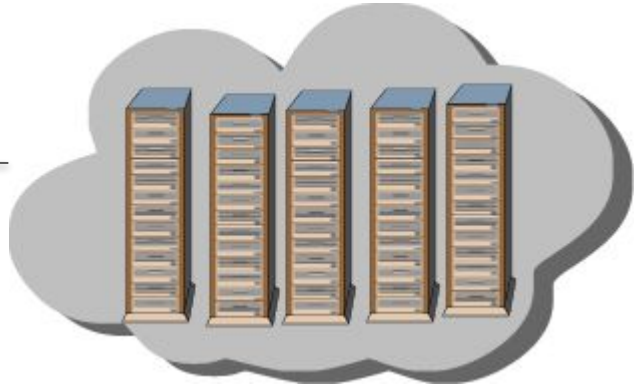
What is Homomorphic Encryption?

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

$Enc_{pk}(x)$

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)

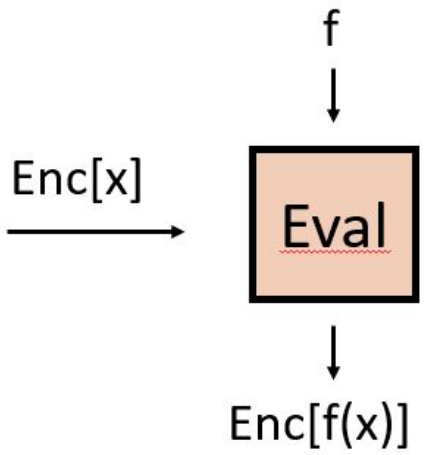
Slide from Shai Halevi: Homomorphic Encryption (Part1):SWHE

Homomorphic Encryption 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-I

- 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**
      - Let Rq be the ring of polynomials
        - All coefficient of the polynomial reduced to mod q
      - According to RLWE: samples of the form (a, a · s + e), it is **difficult** to determine the **secret polynomial s** ∈ Rq, where the polynomial **a** ∈ Rq is sampled **uniformly at random** and the coefficients of the error polynomial **e** are small samples from the **error distribution** χ.

Key Elements of Homomorphic Encryption-II

- A Gaussian Sampler
- Modular Arithmetic
  - Multiplication of Polynomials
    - **Number Theoretic Transform(NTT)**
      - Why not FFT?
        - FFT and inverse-FFT perform arithmetic using real numbers and thus suffer from **approximation errors**.
          - Not desired in cryptographic applications
- Bootstrapping Technique

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.

Knuth Yao Sampling Features

Why use this?

- Number of **random bits required is close to entropy of distribution**- (Near Optimal)

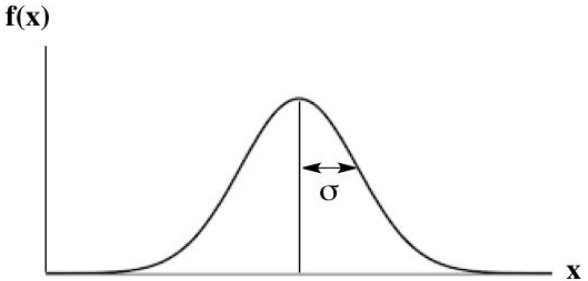
- Knuth Yao proposed a **random walk model for sampling from any non-uniform distribution**.
  - This paper is the **first implementation** of this algorithm
    - Statistical distribution below 2 ^(-90)
      - High precision
      - Large tail bound

- **Optimisation Scope**
  - Simpler approach to traverse the **Discrete Distribution Generating DDG** tree at run time.
  - Column wise storage of probabilities along with **one-step compression** method which results in a near **optimal space** requirement for storage.

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:

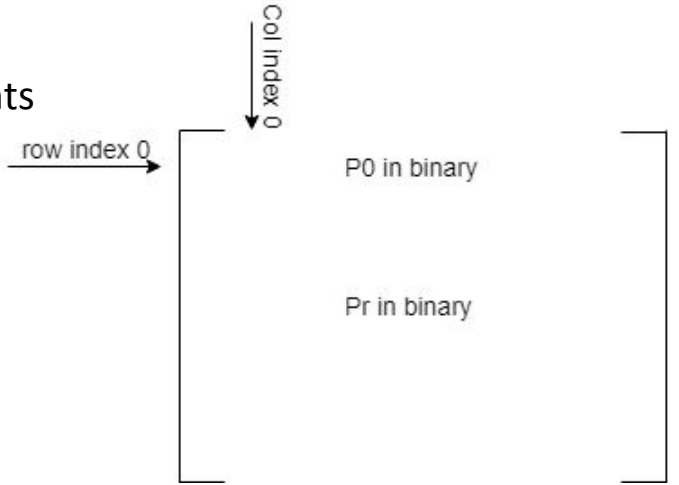
$$Pr(E = z) = \frac{1}{S} e^{-z^2/2\sigma^2} \quad \text{where } S = 1 + 2\sum_{z=1}^{\infty} e^{-z^2/2\sigma^2}$$

- Where S is a **normalisation factor** approximately equal to $\sigma(2\pi)^{0.5}$
- 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.

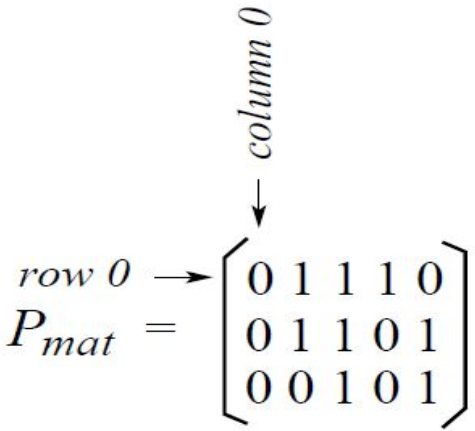
- Let Sample Space for a random variable X consists of n elements
  - 0< **r** <n-1
  - Let p_r be the probability value with respect to **r**
  - We can create a Pmat as follows



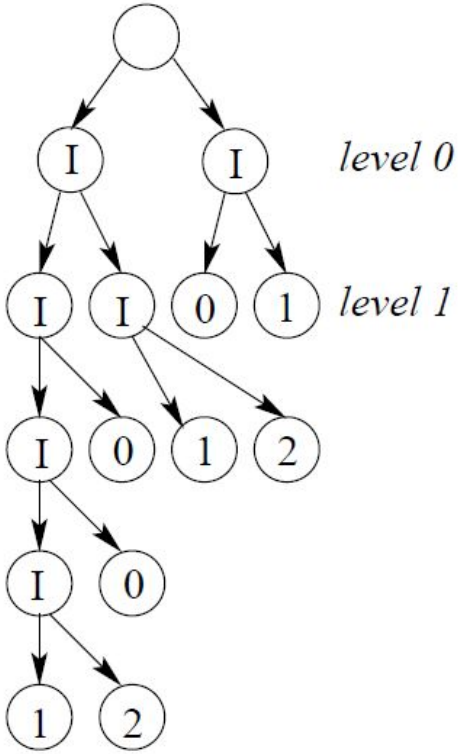
- Using this Pmat we can create a DDG.
  - DDG is a **rooted binary tree**
  - It has two types of nodes
    - **Terminal Nodes**
    - **Intermediate Nodes**

- **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



- The number of Terminal Nodes equals the Hamming Weight of the Column
- Every Intermediate node divides into two node
- Root node always splits into two nodes.



- First Column HW=0
- Second Column HW=2
- Row Index 1 and 0 are 1
- Write the row index in bottom up fashion. Here Right to Left

DDG Traversal

- During the sampling operation
  - Start with the root node
  - Take a random bit (RNG required)
  - Go to the left or right child node say left for random bit 0 and right for 1
  - End of sampling operation when it hits a terminal node.
  - The terminal node gives the row index of the sample in the probability matrix
- Space for DDG is O(nk) where **k is the number of columns (precision)** in the Probability Matrix

DDG Traversal

- In Hardware we need to design the sampler architecture for the **worst case storage requirement** which **makes the implementation costly.**

- ***Optimisation Scope***:-
  - i^{th} level of the DDG can be created from the **$(i-1)^{th}$ level and the i^{th} column of the probability matrix**

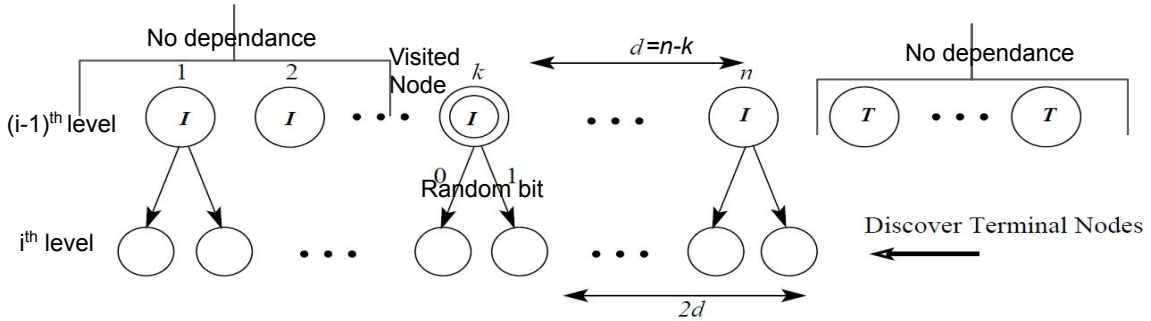
  - So a DDG level can be created during traversal from the previous level and the Pmat.

  - **More Optimisation** just use the visited node say in $(i-1)^{th}$ level and the ith column of the Pmat.

Let's understand with an example

Efficient Implementation of Knuth Yao Algorithm-I

- **Construction of DDG Tree during Sampling**



DDG Property

- All the **intermediate nodes** on the left side and terminal node on the **right side.**
- Number of Terminal nodes in a level say i is the **HW of the Column i in Pmat.**

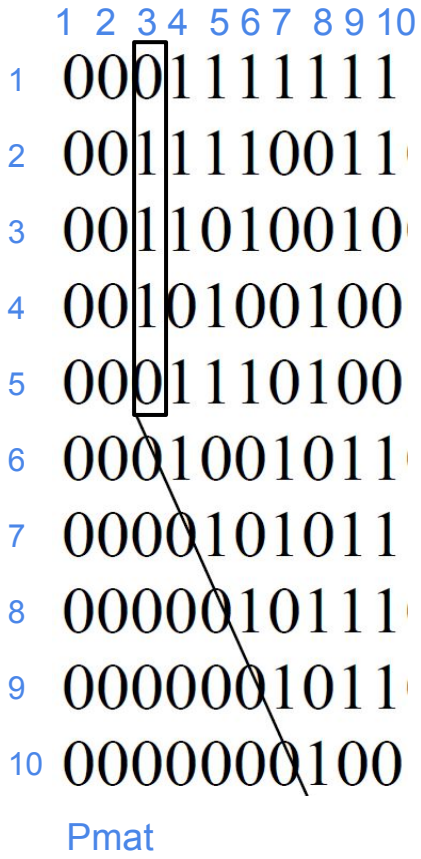
- Now, In the ith level we need to find whether we hit a terminal node or a intermediate node.
  - Scan **the ith column of Pmat**, h be HW of ith column.
  - If random bit =0
    - h>2d+1, we hit a terminal node, otherwise intermediate
  - If random bit =1
    - h>2d, we hit a terminal node, otherwise intermediate
  - If we **hit terminal node return** the row corresponding to the index as the sampling result
  - If we hit an intermediate node, continue

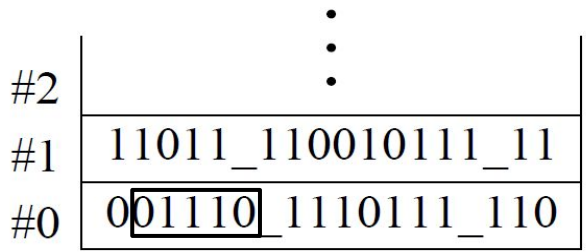
Efficient Implementation of Knuth Yao Algorithm-II

- **Storing the Probability Matrix Efficiently**
  - The Storage depends on **both n (depends on tail bound)** and **c (depends on precision)**.
  - Data is stored in the RAM
    - Data fetching from the RAM increases the computation time

- How do we store data in ROM?
  - As Pmat is accessed Column Wise, store **Pmat Columns in ROM Words**
  - Near the bottom of the Column, **lots of zeros**
    - **Compress the Zeros by storing the Column lengths**
      - The column length is the length of the top portion after which the chunk of bottom zeros start.
      - **Store the differential length** i.e for the next column store the difference in the Column length of the present and next column.
- How?
  - Let’s understand with example

Efficient Implementation of Knuth Yao Algorithm-III





First two ROM words

- Consider the #0 and #1 ROM Word.
  - (0)01110 is the Column 3 from bottom up above the line
  - (1)110111 is the Column 4 from bottom up above the line
    - (1) signifies that the column length difference between the Column 3 and Column 4.
  - We need **not to store the "_" in the ROM**, just initialising the initial column length to length of first non-zero column will work out.

Hardware Architecture-I

The component involved in the Hardware Architecture are as follows:

1. **ROM** stores the probability matrix (compressed) as discussed
   a. The Probabilities are accessed through ROM Counter
      i. Initially the Counter is Cleared and incremented one by one to fetch data from higher ROM locations.

2. Data from the ROM is stored in a **SCAN Register**, which is a left shift register
   a. A counter number of bits scanned from the ROM when this value reaches the WORD SIZE of the ROM.

3. A **Random Bit generator**
   a. Random bit generator based on the **approach of Golic** is use in the Paper (TBD)
   b. On an average only 5 bits are required, thus the random bit generator can be slow.

Hardware Architecture-II

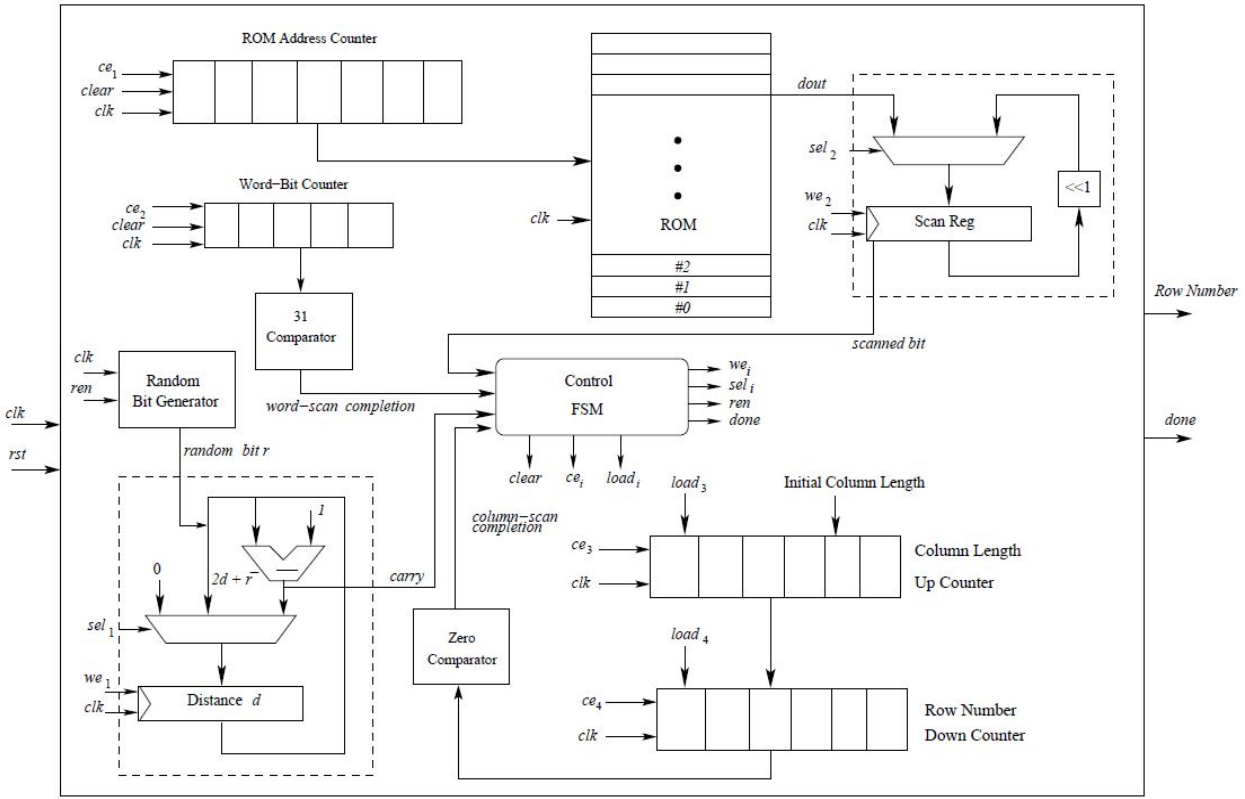
1. Column length **Up counter** stores the column lengths of different columns of Pmat.

2. A **Row Down counter** during column scanning i.e HW.
   a. At the start this value is set to the Column length and then decremented one by one to reach zero.

3. During the random walk the "**d**" is stored in the **distance register.**
   a. It is updated to 2d or 2d+1 depending on the random bit.
   b. Later each detection of terminal node i.e the 1 in the column decrements this by 1 .
   c. Any moment when d<0. Terminate

4. A **FSM Circuit** is used to control the various muxes, Counters present the architecture.

Hardware Architecture



Problem in Knuth Yao Algorithm

- **Problems**
  - If the sum of all the probabilities for a certain tail bound is **not equal to 1**. It is **unsure** whether the Knuth Yao Algo. **will hit a terminal node**.

- **Solution**
  - We use o**ne extra sample point valued as (1-Psum)** which if hit is discarded and the probability of such event is **very very low**.
  - This sample point is regarded as out of range event.

- **Further Improvement in Statistical Distance -Idea [My Inference]**
  - The statistical distance can be increased i.e the **sampled distribution can be matched to the ideal distribution** using a **different floating point representation for the Probabilities at the tail region which are very less.**
    - **Can we use this?**

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**

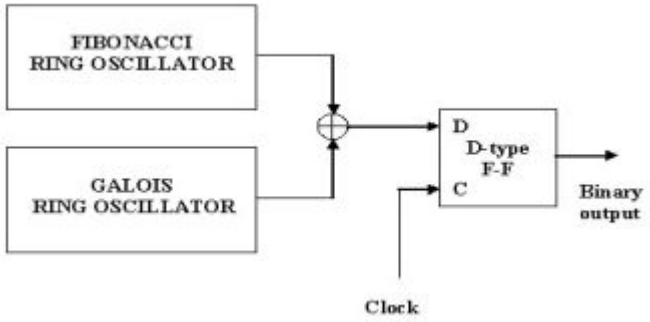


Fig: Digital RNG generator

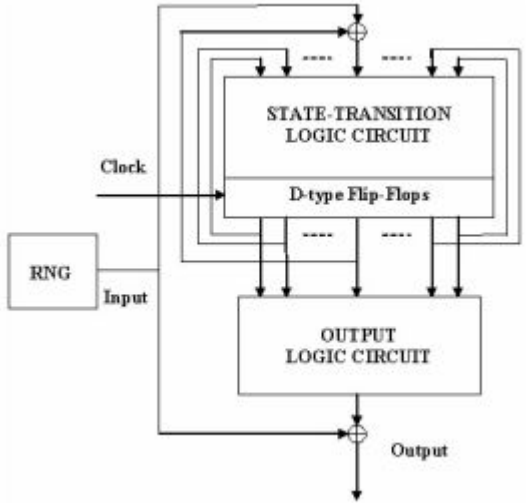


Fig: Generic Post-Processing Circuit

Golic's RNG Scheme-II

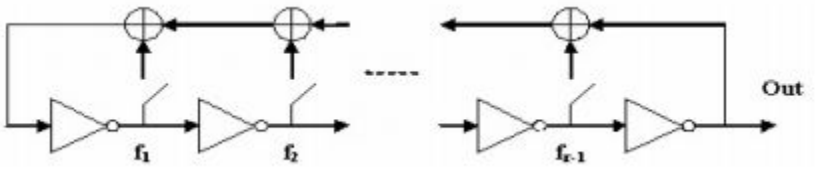


Fig: Fibonacci Ring Oscillator

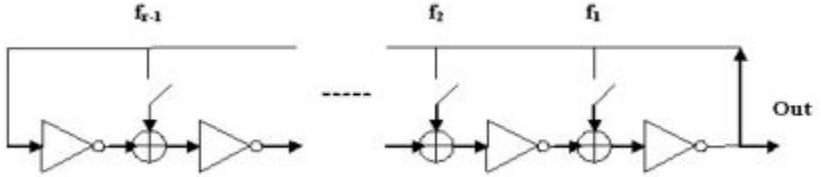


Fig: Galois Ring Oscillator

- 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.

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 Logic	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.

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

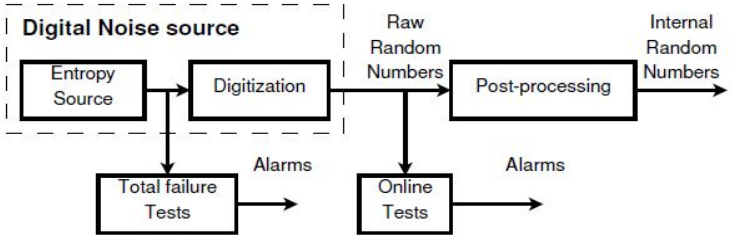


Fig:Generic TRNG Architecture

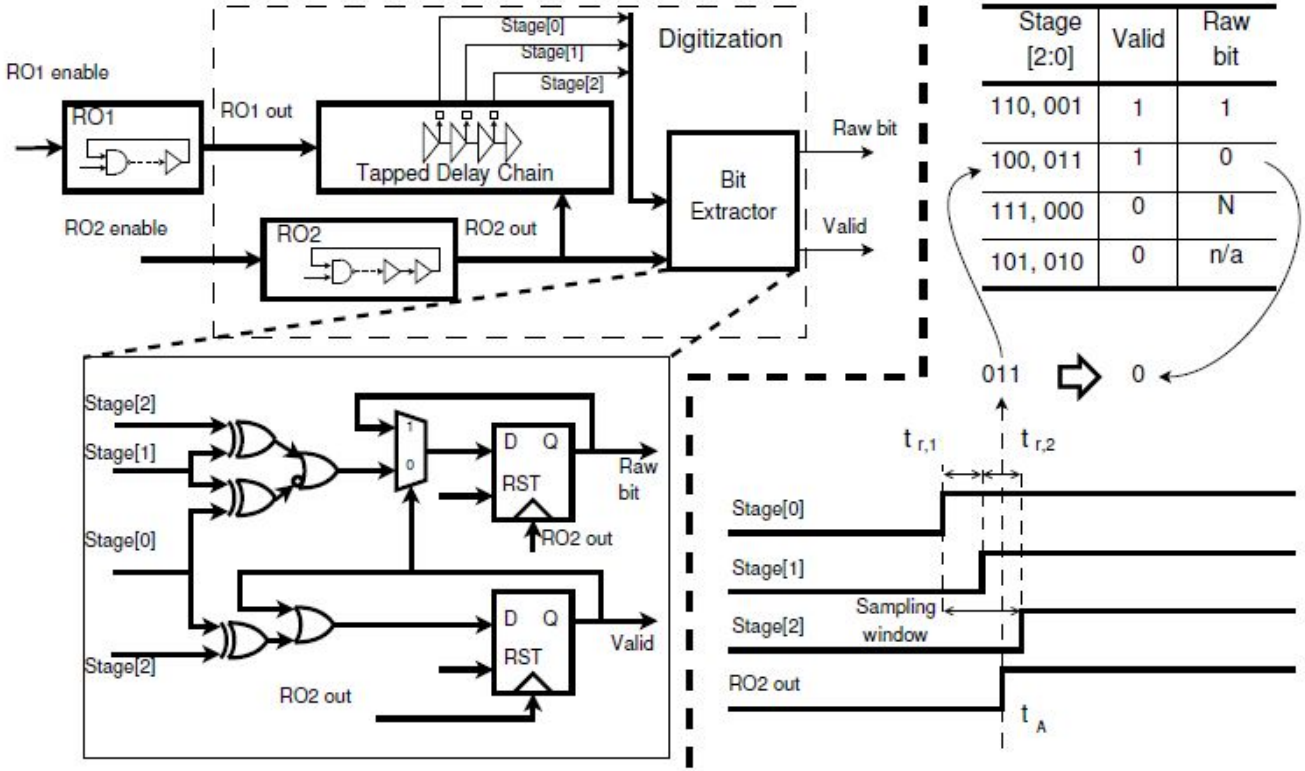
ES-TRNG Important Features

- Throughput of **1.15 Mbps with 0.997 bits of Shannon entropy,** only 1**0 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 **stochastical 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

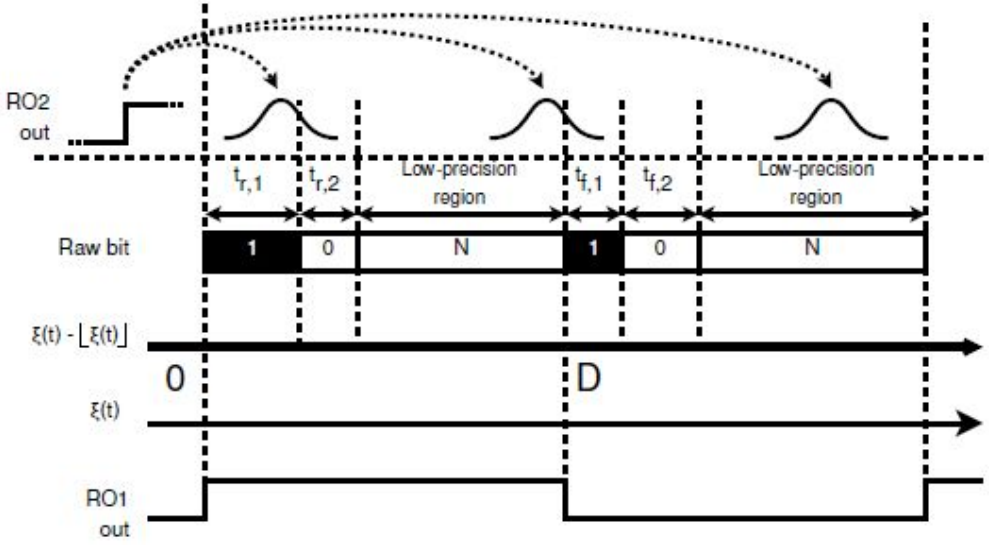


ES-TRNG Parameters

- **Platform Parameter**- the variance in a white noise jitter accumulated during the measurement time tm.
  - Calculated Experimentally
  - Specific to Implementation
  - It is used for the **stochastical modelling.**
  - In simple terms the delay of the logic gates

- **Design Parameters** are Period T01 and T02 of the ring oscillator (R01) and (RO2) respectively.
  - The sampling frequency
  - Jitter accumulation time
  - System Clock

ES-TRNG Techniques-I



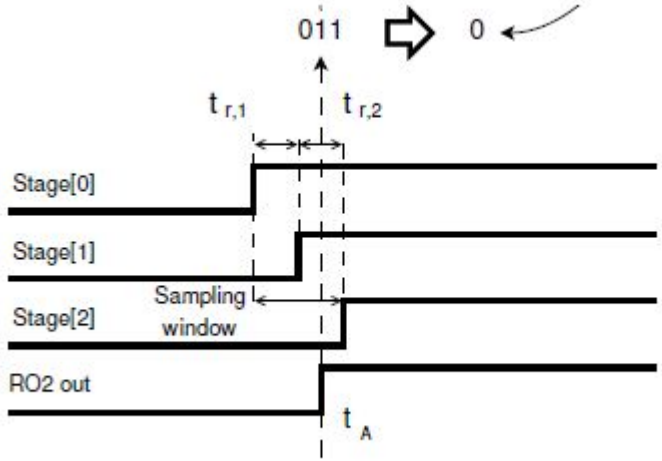


Fig:Variable Precision Phase Encoding

- This result in compact implementation
- Shorter Jitter Accumulation Time
- Entropy is reduced since the low precision is used.

ES-TRNG- Technique II

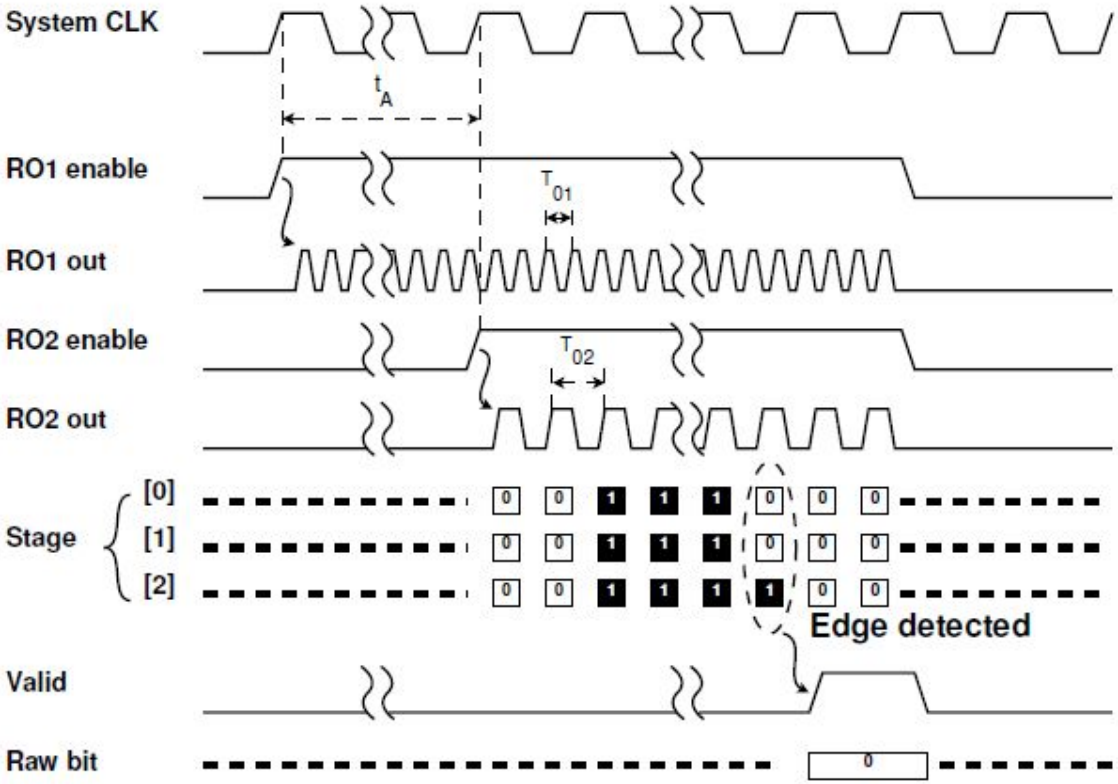
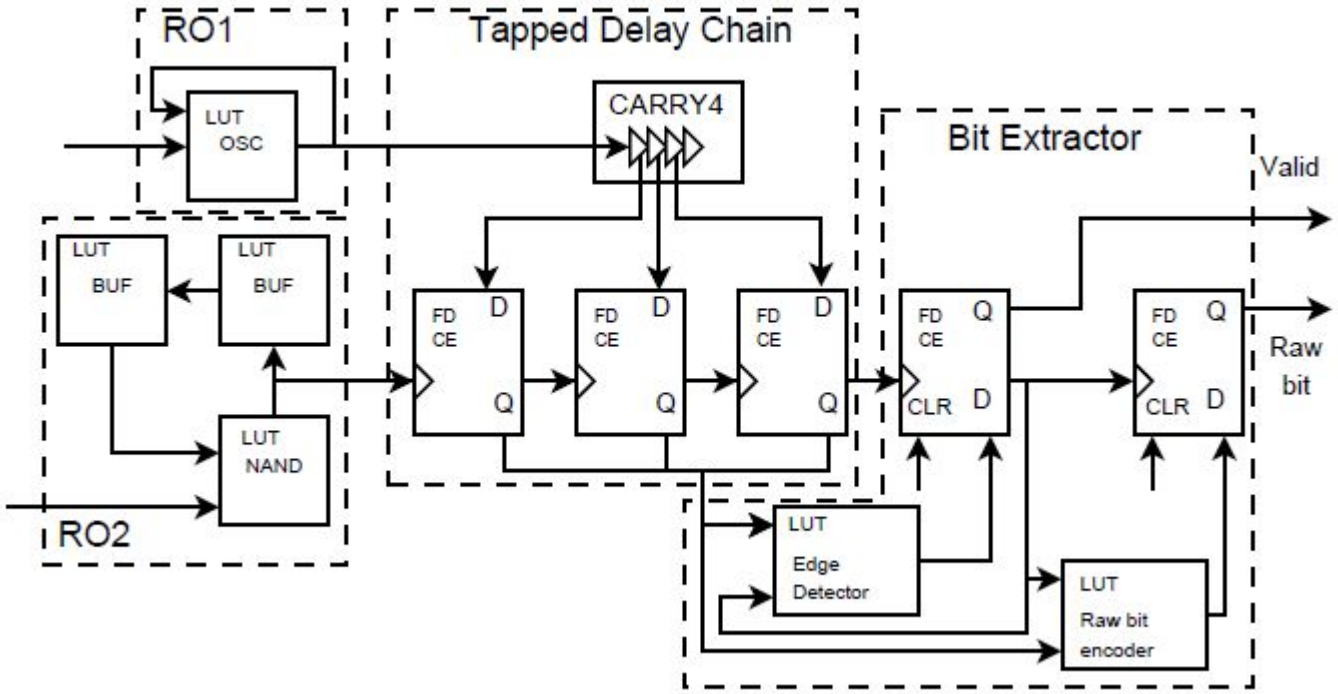


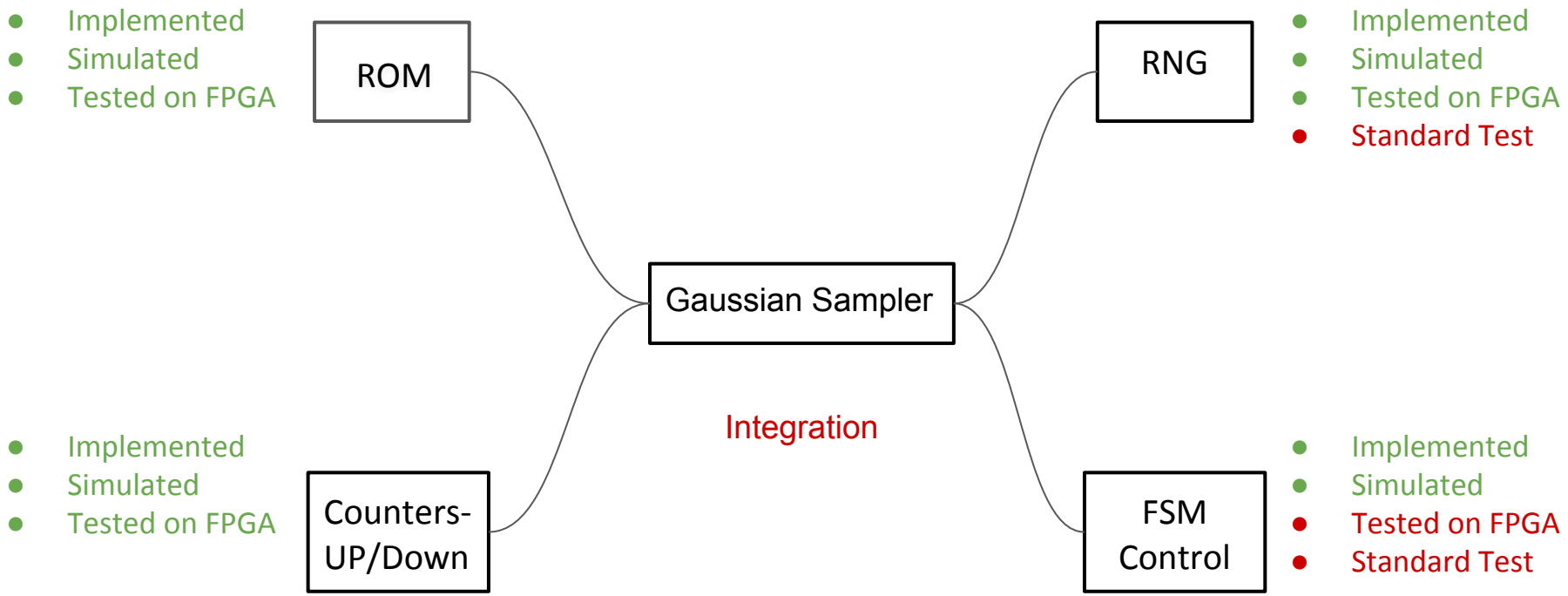
Fig: Repetitive Sampling

- Improving the throughput.
- t_A 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.

ES-TRNG FPGA Architecture



Present Status



Conclusion

- Testing the designed ES-RNG output with AIS-31 Standard
- Integration of the submodules of Gaussian Sampler
- Development of additional key elements of the HE scheme
- Reviewing additional HE schemes like THFE for improvement scope

References

- 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

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

- 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.