Published by :
http://www.ijert.org

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 8 Issue 06, June-2019

VLSI Implementation of an Efficient Processing Unit for Fully Homomorphic Encryption

Blessy Cherian
SENSE Department
Vellore Institute of Technology
Vellore, Tamil Nadu, India

***Abstract*--This paper presents an efficient processing unit for Fully Homomorphic Encryption and Decryption hardware design using finite state machine. The encryption unit consist of a three Linear feedback shift registers, a noise correlator and FSM. The decryption unit has combinational blocks. The processing unit consist of a single hardware having an encryption as well as a decryption unit and a cloud computing unit. There are two proposed architectures, one with a normal LFSR and the other with modified Fibonacci LFSR. As the two architecture is compared, the processing unit with modified Fibonaaci LFSR using FSM has a reduction in the power as well as area.**

Keywords—Fully Homomorphic Encryption;Cloud Computing; Processing Unit; LFSR; Finite State Machine

I. INTRODUCTION

Cloud Computing has raised as an extensive paradigm that has attracted attention in both commercial and educational section. Cloud Computing[8] exists in different names like "outsourcing" as well as "server hosting." The word cloud computing means to store and access data and program over internet instead of storing the data in the hard drive of computer or in large capacity systems. But the poor performance of processors used, slow Internet connections and the exorbitant costs of the materials used, do not allow the use of services and storage spaces. However, recent advances in current technology[8][7] of virtualization, paved the way for these operations with faster processing. This is an attractive solution that can provide low cost storage and processing capabilities for government organizations, hospitals, and small or medium enterprises. It has the advantage of reducing the IT expenses and providing services for the requesting parties through making specialized software and computing resources available. Nevertheless, there are some apprehensions that should be considered by all organization migrating to cloud computing. As the data is transferred to the Cloud, encryption technique is used to secure the operations and the storage of the data. The basic concept is to encrypt the data before sending it to the Cloud provider or the server. The client provides the private key to the server to decrypt data, which might affect the confidentiality and privacy of data stored in the Cloud. A method to execute operations on encrypted data without decrypting them, which will provide the same results after calculations as if we have worked directly on the raw data.

Homomorphic Encryption[21] systems are used to perform operations on encrypted data without knowing the private key which means without decryption, the client is the only holder of the secret key. When we decrypt the result of any operation, it is the same as if we had carried out the calculation on the raw data.

The paper is structured as follows. Section II recapitulate the earlier works in this topic. Section III introduces the Fully homomorphic encryption and different types with equations. Section IV describes the use and architecture of processing unit for homomorphic system. Section V presents the main experimental results collected from hardware synthesis. Section VI concludes the paper with some final remarks.

II. PREVIOUS WORKS

Cloud computing arose as an important case for a large class of applications. Security is a major concern in cloud processing, pointing out the importance of advanced cryptographic techniques like homomorphic encryption, allowing computation to take place on encrypted data on the server side. In specific, this work addresses Fully Homomorphic Encryption (FHE), introduced by Gentry’s[3][1] inspiring work just a few years ago. In 2009 Gentry familiarized a fully homomorphic encryption (FHE) scheme. FHE allows the evaluation of arbitrary functions directly on encrypted data. The Gentry–Halevi scheme was the first software implementation of FHE, but this implementation remains unworkable due to the high latency. An implementation of a variant of the original scheme is proposed by Gentry and Halevi[2]. Their solution, despite various optimizations and small size security parameters, takes more than one second for encrypting a single bit on an Intel Xeon server. Recent software implementations include, open-source library, crypt, is available on-line, while contains an optimized implementation reaching a significant speed-up over the previous solutions. Several research works concerning FHE computing platforms have looked for alternative architectures, mainly GPUs and FPGAs. A FPGA-based accelerator[6] implementing ultralong integer multiplication, the main performance bottleneck in most homomorphic encryption schemes, but this process has been time consuming. The work describes an implementation based on an Altera’s Stratix V FPGA platform. For high performance data path element, which provides high speed architecture by combining the karatsuba operand splitting parallel multiplier with existing Toom cook based modular multiplier[4] and another architecture with FFT based modular multiplications. Generally, modular multiplications is major mission for FHE encryption which consists of modular multiplications which is used to convert plain text to cipher text. Our architecture provides high speed for modular multiplications so large number integers also processed at high speed.

III. FULLY HOMOMORPHIC ENCRYPTION

Homomorphic encryption is a method of encryption that allows calculation on ciphertexts, generating an encrypted result which, when decrypted, matches the result of the operations as if it was performed on the plaintext.

Homomorphic encryption[1] can be used for secure outsourced computation, such as secure cloud computing services, and securely chaining together different services without exposing sensitive data. For example, services from different companies to estimate the tax, the in the case of currency exchange rate. Homomorphic encryption[19] can also be used to create more secure systems such as secure voting systems, collision-resistant hash functions, private set intersection, and private information retrieval schemes. In highly controlled industries, such as healthcare, homomorphic encryption can be used to enable new services by removing privacy blockades inhibiting data sharing. For example, analytics in health care can be hard to use due to medical data privacy concerns, but if the predictive analytics service provider can function on encrypted data in place of these privacy concerns are diminished.

Homomorphic encryption schemes are inherently soft. In terms of malleability, homomorphic encryption schemes have weaker security properties than non-homomorphic schemes. A cryptosystem that supports arbitrary computation on ciphertexts[2][3] is known as fully homomorphic encryption (FHE). Such a scheme enables the creation of programs for any wanted functionality, which can be performed on encrypted inputs to produce an encryption of the output. Since such a program need not decrypt its inputs, it can be run by an untrusted party without revealing its inputs and internal state. Fully homomorphic cryptosystems have great practical implications in the outsourcing of private computations. Mostly, FHE allows for arbitrary computations[9] on encrypted data. Computing on encrypted data means that if a user has a function f and want to get f $(m_1, \ldots, m_n)$ for some inputs $m_1, \ldots, m_n$, it is possible to instead compute on encryptions of these inputs, $c_1, \ldots, c_n$, obtaining a result which decrypts to $f(m_1, \ldots, m_n)$.

An encryption is homomorphic, if, from Enc(a) and Enc(b) it is possible to compute Enc(f (a, b)), where f can be: +, ×, ⊕ and without using the private key. Among the Homomorphic encryption we differentiate, according to the operations that allows to assess on raw data, the additive Homomorphic encryption only additions of the raw data is the Pailler and Goldwasser-Micalli cryptosystems, and the multiplicative Homomorphic encryption only products on raw data is the RSA [14] and El Gamal [9] cryptosystems. E_k is an encryption algorithm with k and D_k is a decryption algorithm with key L, as represented in (1) and (2).

$$D_k(E_k(n)\times E_k(m)) = n\times m \text{ or } Enc(x\otimes y)=Enc(x)\otimes Enc(y) \quad (1)$$

$$D_L(E_L(n)\times E_L(m)) = n\times m \text{ or } Enc(x\oplus y)=Enc(x)\otimes Enc(y) \quad (2)$$

A. *Multiplicative Homomorphic Encryption*

Multiplicative homomorphic encryption is also known as RSA cryptosystem. Suppose x_1 and x_2 are the plaintexts. Then the equation is defined as (3),

$$e_K(x_1)e_K(x_2)=x_1^b x_2^b \bmod n=(x_1x_2)^b \bmod n=e_K(x_1x_2) \quad (3)$$

B. *Multiplicative Homomorphic Encryption*

Additive homomorphic encryption is also known as Paillier Cryptosystem. Suppose x_1 and x_2 are the Plaintext. Then defined as in (4) and (5),

$$e_K(x_1,r_1)e_K(x_2,r_2)=g^{x1}r_1^n.g^{x2}r_2^n \bmod n^2 \quad (4)$$

$$e_K(x_1,r_1)e_K(x_2,r_2) =g^{x1+x2}(r_1r_2)^n \bmod n^2 \quad (5)$$

To perform addition and multiplication on encrypted data stored in the cloud provider, the client must have two different key generators (one for RSA and one for Paillier).

C. *El Gamal Cryptosystem*

El Gamal cryptosystem that is basically a multiplicative homomorphic cryptosystem but by modifying coding mode we can make it additive. El Gamal Cryptosystem performs multiplicative homomorphic encryption propriety Let x_1 and x_2 be plaintexts. Then (7),

$$e_k(x_1, r_1)e_k(x_2, r_2) = \alpha^{r1+r2} \bmod p, (x_1\, x_2)\, \beta^{r1+ r2} \bmod p \quad (7)$$

If we put the plaintext in the exponent, we get the equation as (8),

$$e_k(x, r) = (\alpha^r \bmod p, \alpha x\, \beta^r \bmod p) \quad (8)$$

IV. PROCESSING UNIT FOR CRYPTOGRAPHY

The processing unit[18] is used to process data for effective cost and to avoid hardware. The idea behind this Processing Unit is, many companies need to perform processing operations on the data present but because of the cost effectiveness and to avoid hardware providing place they decide to use third party clouds. The process is, data is encrypted using corresponding encryption algorithm and transmitted to cloud. The cloud will perform operations on encrypted data but the cloud does not have knowledge of the data. The key computed encrypted data is transmitted again to processing unit. After decryption the result is obtained without using complex computational hardware. In the architecture shown in Fig.1 we implemented Gentry's Fully Homomorphic Encryption[2] and Decryption. The Encryption consists of three LFSRs, one noise correlation and one Finite State Machine for Encryption process. The Decryption consist of combinational block for decryption process. The Multiplier consists of one parallel 32x32 multiplier[12][4] for processing the encrypted data one clock cycle one pair of data both addition and multiplication it is also similar like mimic of cloud vendors.

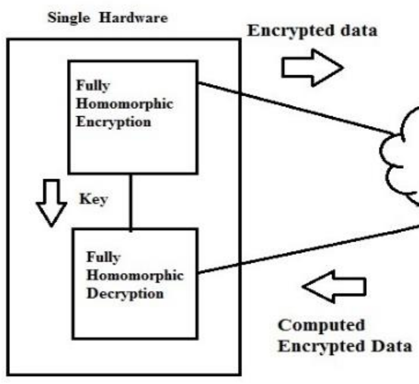
Single Hardware
Encrypted data
Fully Homomorphic Encryption
Key
Cloud Computing (Add,Mult)
Fully Homomorphic Decryption
Computed Encrypted Data

Fig. 1. Architecture of processing unit

A Linear Feedback Shift Register Linear Feedback Shift Register is a sequential shift register with combinational logic that causes it to pseudo randomly cycle through a sequence of binary values. Feedback around an LFSR's shift register comes from selection of points in the register chain and constitutes XOR of these taps to provide taps back into the register. Register bits that do not need an input tap, operate as a standard shift register. It is this feedback that causes the register to loop through repetitive sequences of pseudo random value. The choice of taps determines how many values are there in a given sequence before the sequence repeats. A n-bit Linear Feedback Shift Register (LFSR)[17] is a n-bit length shift register with feedback to its input. The feedback is formed by XOR or XNOR of the outputs of selected stages of the shift register mentioned as taps. The input to least significant bit. The linear part of the term LFSR derives from the fact that XOR and XNOR are XOR and XNOR are linear functions. An LFSR will produce a pseudorandom sequence of length (2n – 1) states. An LFSR is of maximal length when the sequence generates passes through all possible 2 passes through all possible 2n-1 values. The LFSR sequence depends on the seed value. In an LFSR, the bits contained in selected positions in In an LFSR, the bits contained in selected positions in the shift register are combined in some sort of function the shift register are combined in some sort of function and the result is fed back into the register's input bit. By definition[16], the selected bit values are collected before the register is clocked before the register is clocked and result of the feedback function is inserted, filling the position that is emptied as a during the shift. The implemented LFSR typically uses a one-to-many structure, rather than a many-to-one structure, shortest clock-to-clock delay path.

Fibonacci LFSR: In the fig.2 the Fibonacci LFSR with the rightmost bit of the LFSR is called the output bit. The bit positions that affect the next state are called the taps. The taps are XOR sequentially with the output bit and then fed back into the leftmost bit. The sequence of bits in the rightmost position is called the output stream. The bits in the LFSR state that influence the input are called taps.

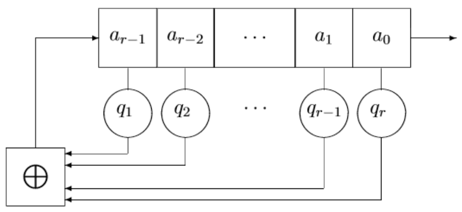

Fig. 2. Fibonacci LFSR

A maximum length LFSR produces an m sequence, unless it contains all zeros, in which case it will never change. As an alternative to the XOR based feedback in an LFSR, XNOR can be used. This function is not strictly linear, but it results in an equivalent polynomial counter whose state is the complement of the state of an LFSR. A state which has all ones is illegal when using an XNOR feedback. In the same way as a state with all zeroes is illegal when using XOR. This state is considered illegal because the counter remains locked up in this state. The arrangement of taps for feedback in an LFSR can be expressed in finite field arithmetic as a polynomial mod 2. This means that the coefficients of the polynomial must be 1 or 0. This is known as feedback polynomial or reciprocal characteristic polynomial. There can be more than one maximum length sequence for a LFSR length. Also, once one maximum length tap sequence has been found, another automatically follows. If the tap sequence in an n-bit LFSR is [n, S, T, U, 0], where the 0 corresponds to the x0 = 1 term, then the corresponding "mirror" sequence is [n, n − U, n − T, n − S, 0].

Galois LFSR: This kind of LFSR is named after the French mathematician Evariste Galois, an LFSR in Galois configuration, which is also known as modular or internal XOR or one to many LFSR, is an alternate structure that can generate the same output stream as a conventional LFSR. In the Galois configuration as shown in Fig 3, when the design is clocked, bits that are not taps are shifted one position to the right unchanged. On the other hand, are XOR with the output bit before they are stored in the next position. The new output bit is the next input bit. The effect of this is that when the output bit is zero, all the bits in the register shift to the right unchanged, and the input bit becomes zero. When the output bit is one, the bits in the tap positions all flip and then the whole register is shifted to the right and the input bit becomes 1. For getting the same output stream, the order of the taps is the counterpart of the order for the conventional LFSR else the stream will be in reverse. The internal state of the LFSR is not certainly the same. The Galois register has the same output stream as the Fibonacci register. A time offset exists between the streams, so a different start point will be needed to get the same output each cycle. Galois LFSRs do not concatenate every tap to produce the new input. Therefore, it is possible for each tap to be calculated in parallel, snowballing the speed of execution. In implementation of an LFSR in case of software, the Galois form is more efficient.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 8 Issue 06, June-2019

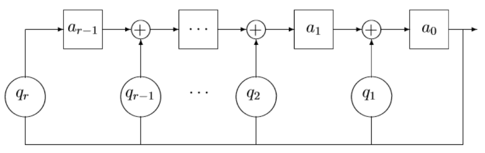

Fig. 3. Galois LFSR

In the architecture, for generation Random numbers for p, q and r we need a LFSR here we are using LFSR with certain seed values. Each p, r_1, r_2, q_1, q_2, we provide different seeds. The value of q_1 and q_2 must be greater than p and also it should be multiples of p, now q_1 and q_2 will be bigger than p for reduction of hardware utilization we implemented a 4 bit LFSR for q_1 and q_2 and multiplying with p so hardware utilization is reduced. In Galois LFSR the critical path is very less compared to Fibonacci LFSR. Extra hardware is used to check whether p is odd or not. If odd, we will pass the values to correlator logic for further processing.

A. Correlator Logic

The equation of FHE[4] before the noise is adding to bits and key, the hacker can reverse it such as by finding GCD of value to decrypt the data so noise is helpful in security scenarios but as multiplications and additions on cloud computing increases noise. Hence, while decrypting we get incorrect data. To avoid this the noise value has to correlated with p values. If the condition satisfies then it will sent to encryption combinational block for encryption of data. Therefore, p value must be as given in (9),

$$p>=(2*r_1+b_1)*(2*r_2+b_2) \tag{9}$$

whereas p is key r_1 and r_2 is noise for pair of cipher text and b_1 and b_2 is corresponding bit value for data pair.

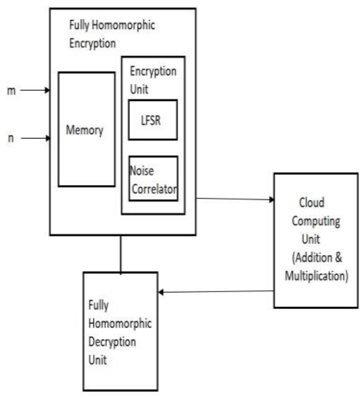
Fully Homomorphic Encryption
m
n
Memory
Encryption Unit
LFSR
Noise Correlator
Cloud Computing Unit (Addition & Multiplication)
Fully Homomorphic Decryption Unit

Fig. 4. Cryptography Processing unit using FHE

B. Encryption Combinational Block

The obtained value of p, r_1, r_2, q_1, q_2 after certain checking and processing. Encryption is to be performed for bits of data. This data is to be encrypted and transferred from memory SRAM using memory controller as in Fig.4. To process four pairs of data, it needs four cycle because the process is performed using single hardware at latency of 4 clock cycles. After encryption the data is transmitted to Multiplication and Addition unit which is mimic of cloud and it is a 32 X 32 multiplier[12] and adder parallel unit. The Memory unit consists of memory which will get input n bits for each clock cycle and store it in memory then it was used by Encryption FSM system for encrypting the data.

Here in Fig.5 the reset block is first activated when we need to clear the data in queue or clear old data because FSM is state machine and load cycle data is loaded from memory and necessary noise and random variable are generated in that cycle and 4 bit of data each bit is encrypted and each cycle using Fully Homomorphic Encryption and data is sent to cloud.

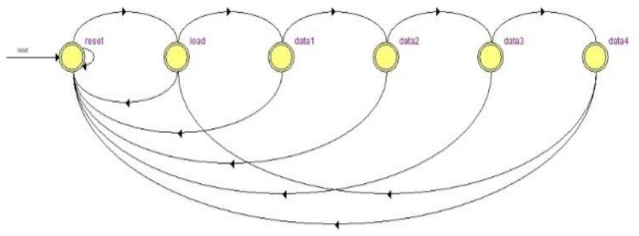

Fig. 5. Encryption using FSM

C. FSM Fibonacci LFSR

The traditional LFSR is considered, having reset signal in synchronous with clock, In the traditional Fibonacci LFSR, a condition is maintained to bypass the stuck at zero state[20]. The state prior to zero state is found by running classical Fibonacci LFSR that is without any modification. This method ensures that all the 256 states are achieved without leading to stuck at 0 states.

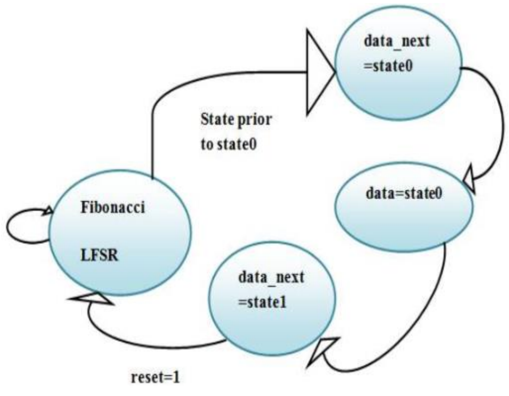

Fig. 6. State Machine for Fibonacci LFSR

In Fig.6, if the reset is 0, then the output of LFSR is assigned to be state 0 that is the output is zero. According to the traditional LFSR if it enters state 0 then it will be stuck there. So next state is assigned to be state 1 that is the output is forced to 1. As long as the reset is zero, the LFSR output will be 0 and next state or output expected from LFSR will be 1. Once the reset is made high, it works as per the polynomial equation described for Fibonacci LFSR. The Fibonacci LFSR polynomial equation is designed in such a way that it skips state 0 which causes the LFSR to enter the stuck state. Therefore, it is required to know the state prior to state 0 and forcefully assign it to some other state. The algorithm for modified Fibonacci LFSR is given as,

i. Declare the inputs for LFSR clock, reset and output data. Also declare an intermediate signal data_next. Both data and data_next are of 8 bit in size.

ii. Check if data is state prior to state 0. If so, then assign next state as state 0. In a classical LFSR this state would be skipped.

iii. If result is false in step ii, then check if data is state 0 if so, then assign data_next as state 1 else perform as per polynomial equation of Fibonacci LFSR.

iv. If positive edge of the clock and reset 0 has occurred and are synchronised, check if reset is zero then assign data to state 0 else assign data to data_next which is calculated from step ii and step iii.

V. EXPERIMENTAL RESULT

The design of the processing unit for encryption as well as decryption was implemented using Verilog. The FHE ASIC was synthesized for 90-nm technology and 32-nm technology, using the Synopsys Design Compiler. The table above shows the power report and the area report of the processing unit using normal LFSR and by Fibonacci LFSR using FSM. The power and the area of modified Fibonacci LFSR is less compared to normal LFSR as in table 1 and table 2. The modified Fibonacci LFSR has a reduced power and area as the hardware use is less as compared to a traditional Fibonacci LFSR because it uses finite state machine. Since, the processing unit consist of three LFSRs the total power and area of the whole system is reduced.

TABLE I. TABLE PROCESSING UNIT WITH 90NM TECHNOLOGY

Processing Unit with	**Power (µm)**	**Area (µm^2)**
LFSR	2.6206e+03	140024
FSM Fibonacci LFSR	1.558e+03	57365

TABLE II. TABLE PROCESSING UNIT WITH 32NM TECHNOLOGY

Processing Unit with	**Power (µm)**	**Area (µm^2)**
LFSR	2.1469e+03	42615.5
FSM Fibonacci LFSR	1.0253e+03	17699.55

VI. CONCLUSION

In this paper, an efficient processing unit for encryption and decryption of homomorphic cryptosystem is implemented. The encryption unit consist of three LFSR, a noise correlator and FSM while the decryption unit consist of combinational blocks. The design is implemented using verilog and synthesized for 90-nm technology as well as 32-nm technology. Experimental results showed that the system of processing unit has a reduction in power and area. The modified Fibonacci LFSR is more efficient as compared to normal LFSR.

REFERENCES

[1] C. Gentry, “Fully homomorphic encryption using ideal lattices,” in Proc. 41st Annu. ACM Symp. Theory Comput., Jun. 2009, pp. 169–178.

[2] C. Gentry and S. Halevi, “Implementing Gentry’s fully-homomorphic encryption scheme,” in Advances Cryptology–EUROCRYPT (Lecture Notes in Computer Science). New York, NY, USA: Springer-Verlag, 2011, pp. 129–148

[3] “A fully homomorphic encryption scheme”, C.Gentry, Ph.D. dissertation, Stanford University, 2009.

[4] Ravi S, Ajith krishna R, and Harish M Kittur, “High Performance Datapath Element for Fully Homomorphic Encryption.”

[5] “VLSI Design of a Large-Number Multiplier for Fully Homomorphic Encryption”, Wei Wang, Xinming Huang.

[6] “Securing the Cloud with Reconfigurable Computing: An FPGA Accelerator for Homomorphic Encryption”,Alessandro Cilardo and Domenico Argenziano.

[7] Mahs TEBAA, Said EL HAJII, “ Secure Cloud Computing through Homomorphic Encryption.”.

[8] Youssef Gahi, Mouhcine Guennoun, Khalil El-Khatib, “ A secure database System Using Homomorphic Encryption Schemes”.

[9] Frederik Armknecht, Colin Boyd, Christopher Carr, Kristian Gjøsteen , Angela J¨aschke , Christian A. Reuter, and Martin Strand, “A Guide to Fully Homomorphic Encryption”.

[10] Akhila K, Karuna N, Kavya C, Yasha Jyothi M Shirur,” Design and Implementation of Power Efficient Linear Feedback Shift Register for BIST using Verilog”.

[11] “FPGA Implementation of a Large-Number Multiplier for Fully Homomorphic Encryption”,Wei Wang and Xinming Huang.

[12] M.-D. Shieh, J.-H. Chen, H.-H. Wu, and W.-C. Lin, “A new modular exponentiation architecture for efficient design of RSA cryptosystem,” IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 16, no. 9, pp. 1151–1161, Sep. 2008.

[13] Selvaraj Ravi,”A Performance Comparsion on of Low Power LFSR”.

[14] Shiv Dutta Mishra, Prof. Anurag Shrivastav,” Advance Parallel LFSR for Cryptography”.

[15] Praveen J, M N Shanmukhaswamy,” Power Reduction Technique in LFSR using Modified Control Logic for VLSI Circuit”.

[16] Mark Goresky, Andrew Klapper,”Fibonacci and Galois Representations of Feedback with Carry Shift Registers”.

[17] Praveen, J. & Shanmukhaswamy, M. (2012), ‘Power reduction technique in lfsr using modified control logic for vlsi circuit’, International Journal of Computer Applications 975, 8887.

[18] Barrett, P. (1986), Implementing the rivest shamir and adleman public key encryption algorithm on a standard digital signal processor, in ‘Conference on the Theory and Application of Cryptographic Techniques’, Springer, pp. 311–323.

[19] Cheon, J. H. & Kim, J. (2015), ‘A hybrid scheme of public-key encryption and somewhat homomorphic encryption’, IEEE transactions on information forensics and security 10(5), 1052–1063.

[20] Singh, B., Khosla, A. & Bindra, S. (2009), Power optimization of linear feedback shift register (lfsr) for low power bist, in ‘2009 IEEE International Advance Computing Conference’, IEEE, pp. 311–314.

[21] Singh, J. & Kaur, P. (2016), Digital image watermarking of homomorphic encrypted images: A review, in ‘2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)’, IEEE, pp. 1790–1793.