IEEE Access
Multidisciplinary : Rapid Review : Open Access Journal

Received 21 November 2022, accepted 11 December 2022, date of publication 15 December 2022, date of current version 21 December 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3229892

RESEARCH ARTICLE

Design of Novel Hardware Architecture for Fully Homomorphic Encryption Algorithms in FPGA for Real-Time Data in Cloud Computing

SAGARIKA BEHERA, (Member, IEEE), AND JHANSI RANI PRATHURI, (Member, IEEE)
CMR Institute of Technology, Bengaluru, Karnataka 560037, India
Corresponding author: Sagarika Behera (sagarika.b@cmrit.ac.in)

ABSTRACT In the cloud computing environment, the data is encrypted using a variety of cryptographic techniques before being sent to the cloud. But in this method, if someone wants to do some operation on the ciphertext, then the client needs to share the secret key to decrypt the ciphertext. So, there is a chance of data leakage. Now more research is being carried out on a fully homomorphic encryption scheme (FHE) to maintain the privacy and secrecy of user's data in the cloud environment. Calculations can be performed on the ciphertext in this encryption technique without the need for decryption. The user will use a private key to decode the result into plain text format. But this method takes more time if it is implemented in software. So, it is not feasible to implement FHE on real-time data. To overcome these difficulties, we have developed a new hardware architecture for Brakerski, Vaikuntanathan (BV) fully homomorphic encryption scheme using Field Programmable Gate Array (FPGA). In this work, we have carried out an extensive simulation of the proposed design in MATLAB, and Python to validate the encryption and decryption algorithm. Further, we have simulated the design architecture of the BV scheme in the Questasim simulator and implemented it in an Intel FPGA using the Quartus tool. In this research work, we found that the proposed architecture takes 0.33 sec for key generation, 0.851 sec for encryption, and 72us for decryption when it is implemented using 32 butter-fly-based Number Theoretic Transfer (NTT) and consumes less FPGA hardware resources.

INDEX TERMS Cloud computing, FHE, FPGA, learning with error (LWE), number theoretic transfer (NTT).

I. INTRODUCTION

There is an increase in demand and use of cloud computing for various applications. In today's digital era, cloud computing has a crucial role in storing data in the cloud and executing various operations on that data. To ensure data security in a cloud computing environment, data is encrypted using various cryptographic algorithms before it is stored in the cloud environment. The hardness of all these popular public key cryptographic (PKC) systems is based on the discrete logarithms problem and integer factorization. These challenges are difficult for conventional computers to break the key, but they are simple for quantum computers to solve. As reported in the literature, quantum computing can solve the factoring problem using Shor's algorithm [35]. There is a need for a new type of cryptographic algorithm or mathematical problem which is difficult for quantum computers to solve and thus it will be secure.

Other kinds of PKC systems that are resistant to quantum fluctuations and not easily broken by quantum computers exist. A few examples include code-based, lattice-based, isogeny-based, hash-based, and multivariate systems. Other standard bodies are evaluating and looking for efficient algorithms from these families. Learning With Errors (LWE) fully homomorphic encryption scheme [1], [2], [3], [4], [5] is a lattice-based cryptographic scheme.

Computing operations can be made on the encrypted data using a fully homomorphic encryption scheme (FHE) without first having to decrypt it. More researchers are working in this developing field of study and are showing interest in

The associate editor coordinating the review of this manuscript and approving it for publication was Agostino Forestiero.

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License.
For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/

this area because it is a feasible solution for cloud security. When cloud users store data in a third-party server, the safety and privacy of personal data are of large concern. In the FHE method, there is no need to share the secret key with a third-party. Craig Gentry in the year 2009, proposed the lattice-based FHE explained in his Ph.D. thesis [6]. Since then, many cryptographers and researchers [1], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], [25] are working on this to enhance FHE performance. There are some interesting applications of FHE in various fields given in the paper [26], [27], [28], [36].

Homomorphic encryption schemes have been implemented practically in software and hardware. But software implementation is very slow in practical scenarios. So, implementing FHE in real-time speed becomes the main limiting factor. We demonstrate the FPGA implementation of the LWE-based BV FHE algorithm [1] in this study.

A. RELEVANT WORK

We are going to present some instances where homomorphic encryption algorithms are implemented in software, Graphics Processing Unit (GPU), FPGA, and the time taken to execute it. On a large machine with a number theory library (NTL) as the underlying software, it takes more than 36 hours to assess a full circuit using a leveled homomorphic encryption that can analyze an Advanced Encryption Standard (AES)-128 circuit [29]. In another scenario using Single Instruction, Multiple Data (SIMD) technique to evaluate 54 blocks it takes 40 minutes for each block. In that paper authors also claimed that evaluating 720 blocks takes two and a half days.

The first hardware implementation of FHE over integers on the XILINX Virtex-7 FPGA platform [30]. The speed was improved by a factor of 52.42 compared to the software implementation. A 768K-bit multiplier architectural concept is suggested. When compared to the NVIDIA C2050 Graphics Processing Unit, which has 448 cores working at 1.15 GHz, the Stratix-V FPGA's 64K-point finite-field fast Fourier transform (FFT) processor is twice as fast at 100 MHz. The Brakerski, Gentry, Vaikuntanathan (BGV) scheme which is based on the Ring Learning with Error (RLWE) concept is implemented in FPGA [31]. The proposed architecture to improve performance and reduce computational latency is tested on a Virtex UltraScale FPGA platform running at 150MHz. According to the authors, their design architecture for homomorphic encryption and decryption was 4.60 $\times$ 9.49x quicker than the optimized software implementation on an Intel i7 CPU running at 3.1GHz, resulting in a throughput improvement of 1.03 $\times$ 4.64x above the hardware implementation of BGV.

A design architecture is proposed to improve the computational power of Fan-Vercauteren (FV) homomorphic encryption scheme on a heterogeneous platform Arm with FPGA [32], It is evaluated on a single Xilinx Zynq Ultra-Scale with MPSoC ZCU102 Evaluation Kit. In comparison to the software implementation of the FV method on an Intel i5 processor operating at 1.8GHz, the authors claimed that their technique provided a 13x speedup at 200MHZ. Lopez-Alt, Tromer, and Vaikuntanathan (LTV) based somewhat homomorphic encryption (SWHE) schemes [33] were implemented on Xilinx Virtex 7. This multiplied a large polynomial in 6.25 milliseconds, which is 102 times quicker than the software implementation.

The authors of the paper [36] claimed that they have proposed an FPGA-based accelerator for bootstrap FHE. The evaluation was performed on Xilinx Alveo U280 FPGA. It is applied to train a logistic regression model on encrypted data.

B. OUR CONTRIBUTION

We have proposed a novel design architecture in the paper [34] to implement LWE FHE in FPGA. In this paper, we have carried out extensive simulations in MATLAB, and python to validate the algorithms. The new design architecture for FPGA implementation is simulated in questasim software and implementation in the quartus FPGA tool. The speed of execution and the hardware resource utilization have been tabulated. In our investigation, it was found that lattice-based FHE algorithms are mathematically similar and the major computation is the polynomial multiplier. Designed with Intel Agilex FPGA (10 nanometers) with large numbers of digital signal processing (DSP) multipliers, the speed performance of 250 MHz is archived for encryption and decryption algorithms. By using large numbers of butter-fly, the long-length plain text has been encrypted and decrypted in the order of microseconds. A large number of butterflies is recommended for Cloud Computing which demands huge data encryption and faster speed of execution. Similarly, on the other hand, our architecture with a smaller number of butter-fly is tailored for IoT applications that demand less data for encryption and decryption. For IoT kind of data, a lesser number of butter-fly and even smaller density FPGA can be used. For example, for 512 lengths of a polynomial using 1 butter-fly and 32 butter-fly takes 8.8us and 0.6us respectively. The throughput performance of 15x with 32 parallel butter-fly compared to a single butter-fly. So, it is found that 32 butter-fly-based solutions can be used for homomorphic encryption operations in cloud computing.

Outline of the paper. The paper is structured as follows. Section II introduced lattice theory, BV schemes, and number theoretic transform. Section III covers the proposed novel architecture including the details of the digital logic block used. In section IV, the simulation is set up with parameters, and results of MATLAB, python, and questasim are included. This section also includes results and discussions. Finally, the paper is concluded in section V.

II. THEORETICAL BACKGROUND

A. LATTICE THEORY

The lattice theory forms the basis of the fully homomorphic encryption system. Lattice problems are NP-hard problems. The encryption methods based on the lattice principle are challenging for quantum computers to crack. The terms associated with lattice theory will be discussed in this section.

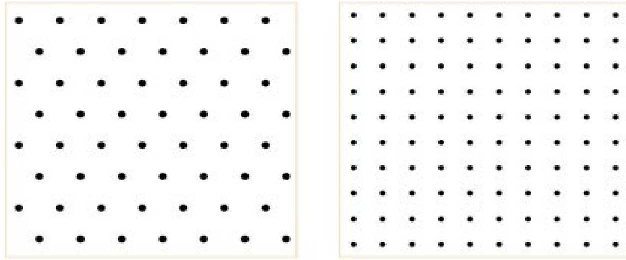

FIGURE 1. Two 2-dimensional lattices in the Euclidean plane.

1) DEFINITION 1 (LATTICE L)

A lattice L of R_n is a discrete subgroup of R_n by definition. Where R_n and Z_n are sets of real numbers and integers respectively. Here integer lattices are considered i.e., $L \subseteq Z_n$. A lattice is any correctly spaced grid of points that extends into the infinite. In Fig.1, two different, 2-dimensional lattices are shown.

2) DEFINITION 2 (BASIS)

A basis B of L is an ordered set

$$B = (b_1, b_2, ..., b_n)$$

such that $L = L(B) = B \cdot Z^n = \sum_1^n c_i b_i : c_i \in Z$.

Here b_i is a column vector. Any point in the grid that makes up the lattice can be recreated using a basis, which is a set of vectors. One thing to keep in mind is that each lattice does not have a single basis. In actuality, it is built on several foundations. Some bases are deemed" good," while others are labeled" bad." A basis is a fundamental finite object that can be stored in a computer's memory as opposed to an infinite grid of dots. The vectors (b_1, b_2) are two bases of the lattice shown in Fig. 2.

3) DEFINITION 3 (SHORTEST VECTOR PROBLEM) (SVP)

Finding a nonzero vector v in lattice L from a given basis. Fig.2 shows an SVP. The basis vector is in blue and the shortest vector is in red.

4) DEFINITION 4 (CLOSEST VECTOR PROBLEM)

Determining the vector v in lattice L that is closest to the target. The closest vector problem is depicted in Fig.3 with the basis represented in blue, the external vector in green, and the closest vector in red.

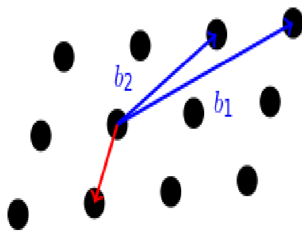

FIGURE 2. Shortest Vector Problem.

B. SHORT INTEGER SOLUTION AND LWE PROBLEM

1) DEFINITION 5 (SHORT INTEGER SOLUTION)

In lattice-based cryptography constructions, the short integer solution (SIS) and ring-SIS problems are two typical difficulties. Problems that are challenging to solve in a random sampling of cases are known as average-case problems. Worst-case complexity is insufficient for cryptography applications, and we must ensure that cryptographic constructions are difficult based on the average case complexity. Mathematically it is represented as $SIS_{n,q,m,\beta}$. Where n is the no. of equations or height of the matrix. q is the finite field of work. m is the no. of variables. β is the short vector length.

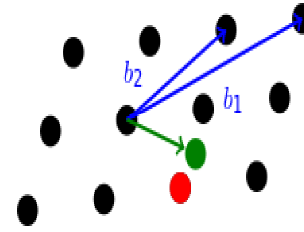

FIGURE 3. Closest Vector Problem.

Given a matrix $A \in Z_q^{n\times m}$ is an $n \times m$ matrix with entries in Z_q that consists of m uniformly random vectors $a_i \in Z_q^n$, search for a nonzero vector $r \in Z^m$ such that $Ar = 0 \mod q$ ($over\ Z_q^n$) and $\|r\| \leq \beta$ Where $A = (\overrightarrow{a_1}, \overrightarrow{a_2}, \ldots, \overrightarrow{a_m})$

2) DEFINITION 6 (LEARNING WITH ERROR PROBLEM)

The LWE problem states that $s \in Z_q^n$ is a secret vector from a set of linear equations on 's' and the goal is to recover the secret vector 's' as specified in [2]. We can find the value of 's' in polynomial time from 'n' equations using Gaussian elimination if there is no error. The different parameters and the LWE problem statement is explained below. Let $n \geq 1$, $q \geq 2$ is the modulus, a vector $s \in Z_q^n$, and an error probability distribution χ on Z_q. Let $A_{s,x}$ be the probability distribution on $Z_q^n \times Z_q$ by choosing a uniform random vector $a \in Z_q^n$ and an error vector $e \in Z_q^n$ and gives the output as $(a, \langle a, s\rangle + e)\ mod\ q$.

3) DEFINITION 7 (MAXIMUM LIKELIHOOD ALGORITHM)

One way of solving the LWE problem is by the maximum likelihood algorithm. Assume 'q' is polynomial and the distribution of error is normal. Approximately after 'n' number of equations, we can demonstrate that the 's' which satisfies approximately all the equations is the correct one.

C. BV FHE SCHEME

We will briefly discuss the BV FHE scheme. This scheme was put forward by Z. Brakerski, V. Vaikuntanathan [1] in the year 2011. In this paper, this scheme is implemented using FPGA. This scheme is established on the LWE presumption. The security of this technique is based on the worst-case difficulty of solving conventional lattice problems. They have introduced two methods to obtain an SWHE and then convert SWHE into FHE. The first step is the re-linearization technique and the second step is dimension-modulus reduction technique. These two methods are explained below.

1) RE-LINEARIZATION TECHNIQUE

Considering the LWE premise, this technique is utilized to generate an SWHE. Encrypting a message bit $m \in \{0, 1\}$

using secret key $s \in Z_q^n$, a random vector $a \in Z_q^n$, error/noise 'e' is selected and 'c' is the output ciphertext, given in (1).

$$c = (a, b = \langle a, s\rangle + 2e + m) \in Z_q^n \times Z_q \tag{1}$$

In the encryption process, two masks are used. One is the hidden mask $\langle a, s\rangle$ and the other one is the even mask $2e$. During the decryption process, these two masks do not interfere with each other. First, we can recompute $\langle a, s\rangle$ and subtract it from 'b', the result will be $2e+m$ (mod q). Since the noise $e \ll q$, then $2e + m$ (mod q) $= 2e + m$. Now to remove the even mask we can calculate $(2e + m)\ mod\ 2$. Below is a detailed explanation of the decryption algorithm, from which we can learn about the homomorphic characteristics of the BV scheme. A ciphertext (a, b) has been given and a linear function $f_{a,b} : Z_q^n \rightarrow Z_q$ is defined in (2).

$$f_{a,b}(x) = b - \langle a, x\rangle(mod\ q) = b - \sum_{i=1}^{n} a[i].x[i] \in Z_q \tag{2}$$

where $x = (x[1], x[2], \ldots, x[n])$ denotes the variables, and (a, b) are the coefficients of the linear equation. Now it is evident that evaluating this function on the secret key 's' and then obtaining the modulo 2 of the result is all that is required to decrypt the ciphertext (a, b).

2) HOMOMORPHIC ADDITION

Let c_1 and c_2 be two ciphertexts. The addition of these two ciphertexts is equivalent to the addition of two linear functions. The result will be another linear function. It is presented in (3).

$$f_{(a+c,b+d)}(x) = f_{a,b}(x) + f_{c,d}(x) \tag{3}$$

The resultant ciphertext will be $(a + c, b + d)$.

3) HOMOMORPHIC MULTIPLICATION

Multiplication of two ciphertexts c_1 and c_2 can be represented by (4).

$$\begin{aligned} f_{a,b}(x).f_{c,d}(x) &= (b - \sum a[i]x[i]).(d - \sum c[i]x[i]) \\ &= h_0 + \sum h_i.x[i] + \sum h_{i,j}.x[i]x[j] \end{aligned} \tag{4}$$

The result will be a polynomial of degree 2 with variables $x = (x[1], x[2], \ldots, x[n])$ and coefficients will be $h_{i,j}$ which can be computed from (a, b) and (c, d). Now the size of the ciphertext increased from $n + 1$ to $n^2/2$. To reduce the size of ciphertext, the re-linearization technique has been applied. With another new secret key 't', all of the linear and quadratic terms under secret keys have been published in this technique. The new form of the linear equation is shown in (5).

$$b_{i,j} = \langle a_{i,j}, t\rangle + 2e_{i,j} + s[i].s[j] \approx \langle a_{i,j}, t\rangle + s[i].s[j] \tag{5}$$

Equation (6) shows the approximate representation of the homomorphic multiplication linear equation using the new secret key 't'.

$$h_0 + \sum h_i(b_i - \langle a_i, t\rangle) + \sum_{i,j} h_{i,j}.(b_{i,j} - \langle a_{i,j}, t\rangle) \tag{6}$$

This resulting ciphertext has maximum $n + 1$ coefficients in the linear equation. When it will be decrypted using the secret key 't', it will give $P.Q$ i.e multiplication of two plaintexts. In this way, it can perform only one multiplication. It can perform L levels of multiplication using a chain of L secret keys.

4) POLYNOMIAL MULTIPLICATION AND NUMBER THEORETIC TRANSFORM

As per the discussion in subsection II-C3, one of the fundamental operations in the BV scheme is the multiplication of two large polynomials. Here we will discuss the Number Theoretic Transform (NTT) method which is used for the multiplication of two large polynomials. It is a generalization method of Discrete Fourier Transform (DFT) to a finite field. NTT allows to perform fast convolutions on large integers without any round-off error. This convolution is very much useful for the multiplication of large polynomials.

Suppose p(x) is the polynomial with a degree less than 'n' and given in (7).

$$p = (p[0], p[1], \ldots, p[n - 1]) \in Z_q^n \tag{7}$$

Let ω be a primitive n^{th} root of unity in Z_q as shown in (8).

$$\omega^n \equiv 1 \mod q \tag{8}$$

The forward transform or the n-point NTT of p(x) is given in (9) for i=0,1,...,n-1.

$$P_i = \sum_{j=0}^{n-1} p[j]\omega^{ij}(\mod q) \tag{9}$$

The (10) can be used to determine the n-point Inverse NTT (INTT) for i=0,1,...,n-1.

$$p_i = n^{-1}\sum_{j=0}^{n-1} P[j]\omega^{-ij}(\mod q) \tag{10}$$

III. FPGA ARCHITECTURE FOR THE BV FHE METHOD

This section includes the proposed hardware architecture to accelerate the BV FHE scheme.

In the BV FHE scheme, most of the operations are performed on the long polynomials. From these operations, polynomial additions and multiplications are the most important operations. The polynomial multiplication takes more time ($O(n^2)$) in real-time data. Software implementation takes more time, so there are research areas to evolve new architecture to implement in programmable hardware such as FPGA to meet the timing need. Here, we have developed a new hardware architecture to accelerate the time of execution in programmable hardware. In the following sections, the new techniques for key generation, encryption, decryption, evaluation of encrypted data, NTT, INTT for polynomial operations, and the hardware design architecture for various blocks have been covered.

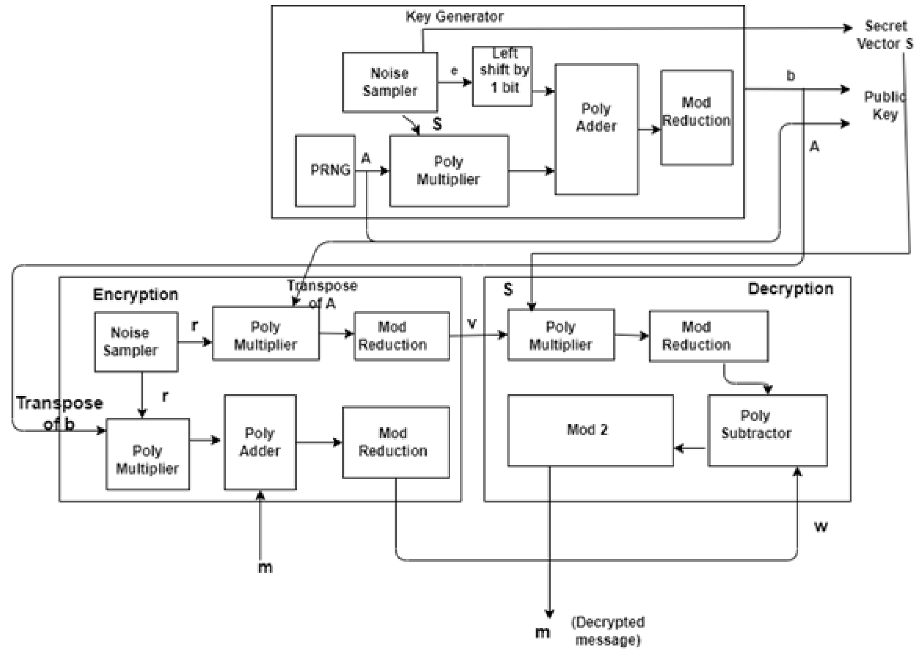

FIGURE 4. FPGA-Based Design Architecture for BV FHE Scheme.

A. NOVEL HARDWARE ARCHITECTURE AND DIGITAL TECHNIQUES

Digital techniques for novel FPGA implementation of the overall architecture of our proposed method are presented in this section. Fig.4 shows the FPGA-based design architecture for the LWE-based BV fully homomorphic encryption scheme. The overall block diagram is divided into three sub-blocks, key generator, encryption, and decryption blocks. The circuit-level design of each block is given in subsequent subsections. The secret vector 's' and small value of noise/error 'e' are generated by the Noise sampler block. This error 'e' is shifted left by 1 bit to get the value of 2e. The public key 'A' is generated randomly by the Pseudo-Random Number Generator (PRNG) block. The value of 'A' and secret vector 's' is multiplied by a polynomial multiplier and the output is added with '$2e$' by a polynomial adder. The modular reduction is applied to the result to get the value of $(As+2e)\ mod\ q$. This value is stored in 'b'. We got the public key as (A, b) and the secret vector 's' as output from the key generator block.

In the encryption block, the noise sampler generates sample values of 0 and 1 and is stored in 'r'. The polynomial multiplier multiplies the transpose of 'A' with 'r' and modular reduction applied on the result to get the value of v, where $v = (A^t r) \mod q$. Transpose of b is multiplied with r and message m is added with it using a polynomial adder. Modular reduction is applied to the result to get the value of 'w', where $w = (b^t r + m)\ mod\ q$. The ciphertext $c = (v, w)$.

The decryption block decrypts the ciphertext 'c' to get back the plain text 'm'. The polynomial multiplier multiplies 'v' and secret vector 's'. Modular reduction is applied to the result and this value is subtracted from 'w' by a polynomial subtractor. So, to get the value of the plaintext 'm', the decryption block performs the operation $m = ((w - vs) \mod q) \mod 2$. Algorithms for all these operations are given in the next section.

TABLE 1. m-Sequence Primitive Polynomial.

m (Degree of polynomial)	Sequence length (N)	Primitive Polynomials
1	1	$p+1$
2	3	p^2+p+1
3	7	p^3+p+1
4	15	p^4+p+1
5	31	p^5+p^2+1
6	63	p^6+p+1
7	127	p^7+p+1
8	255	p^8+p^7+p^2+p+1
9	511	p^9+p^4+1
10	1023	p^{10}+p^3+1
11	2047	p^{11}+p^2+1
12	4095	p^{12}+p^6+p^4+p+1

B. PSEUDO-RANDOM NUMBER GENERATOR

Here we have used m-sequence code to generate PRNG. To generate such sequence 'm' no. of Linear Feedback Shift Registers (LFSR) are used. Table 1 shows the primitive polynomial or generator polynomial which is used to generate m-sequence.

Here we are generating a pseudo-random number of lengths 512. The primitive polynomial used for it is $p^9 + p^4 + 1$.

Fig. 5 shows the detailed circuit diagram to generate 512 bits PRNG. Here we have used 9 no. of LFSR to generate 512 bits of pseudo-random numbers.

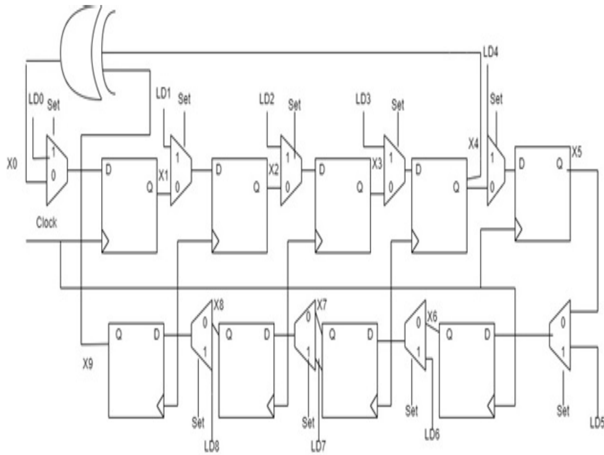

FIGURE 5. Pseudo-Random Number Generator Circuit (512 bits).

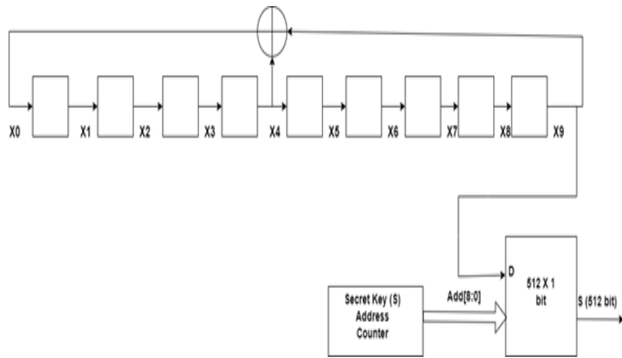

FIGURE 6. Secret Key Generation Block using 9-bit PRNG.

Further, the LFSR has a load option (LD0 to LD8) and by set bit, any arbitrary polynomial can be loaded into the register. It enhances the capability of the PNRG for key generation for applications such as the BV homomorphic encryption scheme.

C. SECRET KEY (S) GENERATION BLOCK USING 9-BIT PRNG

The secret key generation block is implemented as follows. There is a 9-bit shift register and the output x9 is connected to 512×1 RAM which is addressed by a 9-bit counter called a Secret Key(s) Address Counter. In our experimental set up we have taken secret vectors of length 512×1 bits. Fig. 6 shows the block diagram to generate this secret key using 9 LFSRs.

D. PUBLIC KEY (A) GENERATION BLOCK USING 4096-BIT PRNG

To perform hardware simulation of encryption in the BV scheme, we have taken public key 'A' of length 4612 bits. Fig. 7 shows a block diagram to generate 4096 bits (2^{12}) of pseudo-random numbers. Which approximates 4612 bits. The input to the adder shows how the bits are rotated and shifted. The adder output is serially loaded into a 4096×1 RAM which is addressed by a 12 bits counter to generate the public key 'A'. The same hardware is also instantiated to generate Noise/Error(e) as depicted in Fig. 8.

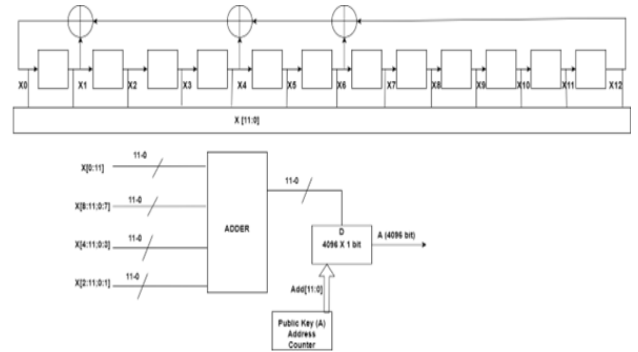

FIGURE 7. Public Key Generation Block using 4096-bit PRNG.

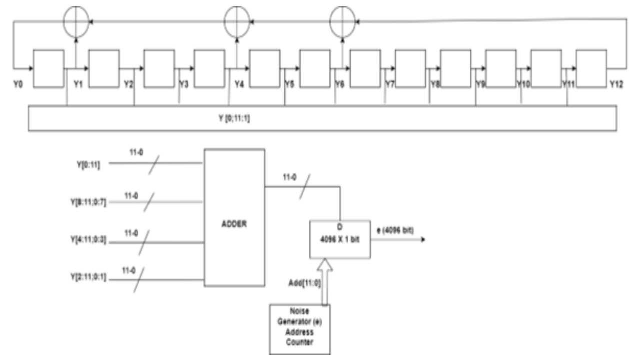

FIGURE 8. Noise Generation Block using 4096-bit PRNG.

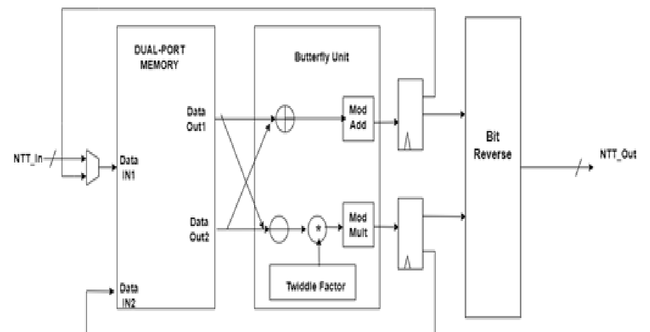

FIGURE 9. Architecture Diagram for NTT Transformation.

E. NOISE/ERROR (E) GENERATION BLOCK USING 4096-BIT PRNG

The size of the error vector or noise vector 'e' is 4612 bits in our experimental setup. Fig. 8 shows the block diagram for generating 4096 bits of the pseudo-random number. It can be repeated to generate 4612 bits.

F. ARCHITECTURE DIAGRAM FOR NTT TRANSFORMATION

Fig. 9 shows a pipelined architecture for Number Theoretic Transform. As shown in the figure, the complete data path is divided into 3 blocks. The blocks are Dual-Port Memory, Butterfly Unit, and Bit Reverse block. In the Dual port memory, input data is stored temporarily. It maintains the polynomial's 'n' coefficients. This Dual port memory helps in reading and writing concurrently. Which helps in the pipeline structure of our proposed architecture.

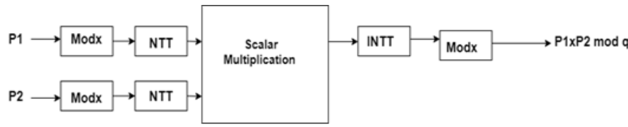

FIGURE 10. Dataflow diagram for NTT-based Multiplier.

The upper data path and lower data path of the butterfly unit correspond to the two input and output data buses of this dual port memory. In a cycle, it can support two reads and two write operations. The butterfly unit performs modular addition and modular multiplication. The Dual port memory receives the output of the Butterfly unit as input to perform the subsequent Butterfly computation. Since input values are in natural order but the index of NTT values is in bit reverse order, at the end bit reverse block performs bit reverse operation to reverse the bits of NTT operation output.

G. DATAFLOW DIAGRAM FOR NTT-BASED MULTIPLIER

The data flow diagram for the NTT-based Multiplier is shown in Fig. 10. P1 and P2 are two polynomials. The Modular operation was applied to these two polynomials before finding the number theoretic transformation. Two polynomials are multiplied by scalar multiplication and inverse number theoretic transformation is applied to it to get the multiplication result. A Modular operation is applied to it to get the final result.

H. ALGORITHM

In this section, we have presented Regev's LWE algorithm, and BV FHE scheme's key generation, encryption, decryption, and evaluation algorithm. The public key pair (A, b) is generated using the key generation algorithm. Using this public key, encryption of plain text message 'm' is performed. The encryption algorithm produces ciphertext as its result. It contains two values, 'v', and 'w'. Each ciphertext has a level tag 'l' attached to it, which shows the multiplicative depth. This is used during the evaluation process. For the first ciphertext, this value will be 0, it gets incremented for each encryption process.

For the polynomial multiplication, the Number Theoretic Transform algorithm and Inverse NTT algorithm are given below.

The Learning with Error algorithm presented by Oded Regev [2] is as hard as a lattice problem which is used in most of the homomorphic encryption schemes for cryptosystems. Most of the cryptographic algorithms which are based on the lattice-based system use either LWE or RLWE. In algorithm 1, we have presented Regev's LWE algorithm. The concept of the LWE algorithm is explained in subsection II-B2.

In the BV scheme, to generate the public key pair (A, b), learning with error concept is used. The secret key 's' is generated by a pseudo-random number generator. The key generation algorithm is presented in algorithm 2.

The encryption algorithm given in algorithm 3 is used to encrypt the plain text message 'm' in the BV scheme. It'll give the ciphertext as output which consists of a pair (v, w) and a level tag 'l' for each ciphertext. Which is required during the evaluation process on the ciphertext. Generation of 'v' and 'w' for plain text 'm', using the public key pair (A, b) given below.

Performing evaluation operations like addition and multiplication on the encrypted data without decryption is the main aim of a homomorphic encryption scheme. These evaluation processes are explained in subsection II-C and algorithm 4 shows the steps of the evaluation process.

The decryption algorithm given in algorithm 5 is the last step to get back the plain text from the ciphertext. After performing the evaluation operation on the ciphertext, the result will be in encrypted form. The user will decrypt the result using secret key 's' to get back the result in the form of plain text. To remove the even mask, mod 2 operation performed.

Polynomial multiplication is required during key generation, encryption, decryption, and evaluation operation.

Algorithm 1 Algorithm for RegevLWE

Input: n, q, m, s, χ

Output: $A \leftarrow \mathbb{Z}_q^{m\times n}$ *Initialization*: Let n, q, m are positive integers $\in \mathbb{N}$. Where $q \geq 2$ is an odd modulus and s is a secret vector, $s \in \mathbb{Z}_q^n$, χ is a probability distribution on $\mathbb{Z}_q$, an error vector $e \in \mathbb{Z}_q$

1: $s \leftarrow \mathbb{Z}_q^n$
2: $e \leftarrow \chi$
3: $b \leftarrow (\langle A, s\rangle + e) \bmod q$
4: **return** A, b

Algorithm 2 Algorithm for Key Generation

Input: n, q, m, s_L, χ

Output: $A \leftarrow \mathbb{Z}_q^{m\times n}$

Initialization: Here the parameters n, q, m, χ have the same meaning as in Algorithm 1. $L \in N$ is an upper bound on the maximum multiplicative depth and each level is associated with a secret vector s_l, where $l \in [L]$. All the $L + 1$ vectors $s_0, s_1, \ldots, s_L \leftarrow Z_q^n$

1: $e \leftarrow \chi^n$
2: **for** $i := 0$ to L **do**
3: $\quad b \leftarrow (\langle A, s_i\rangle + 2e) \bmod q$
4: **end for**
5: **return** public key (A, b)

Algorithm 3 Algorithm for Encryption

Input: A, b, m, r

Output: Cipher text $c \leftarrow (v, w)$

Initialization: The message $m \in GF(2)$ will be encrypted using the public key (A, b), and a sample vector $r \leftarrow \{0, 1\}^m$. For the first ciphertext, the level tag l will be 0.

1: $v \leftarrow (A^T.r)$
2: $w \leftarrow (b^T.r + m)$
3: **return** ciphertext $c \leftarrow ((v, w), l)$

Algorithm 4 Algorithm for Evaluation

Input: The cipher-texts $c_1, c_2, c_3, \ldots, c_t$ where $c_i = ((v_i, w_i), l)$

Output: Resultant output in form of ciphertext

Initialization: Operations on the cipher-texts

1: **for** $i = 1$ to t **do**
2: Homomorphic Addition: $C_i = ((\sum_i v_i, \sum_i w_i), l)$
3: **end for**
4: Find the Homomorphic multiplication of two cipher texts c_1 and c_2
5: Let $c_1 = (v_1, w_1)$ and $c_2 = (v_2, w_2)$ then $c_{mul} = c_1.c_2 = (v_1.v_2, v_1.w_2 + w_1.v_2, w_1.w_2)$
6: **return** $c_{mul} \leftarrow ((v_{mul}, w_{mul}), l + 1)$

Algorithm 5 Algorithm for Decryption

Input: c, s_L

Output: Plain text $m \leftarrow \{0, 1\}^*$

Initialisation: The ciphertext $c \leftarrow ((v, w), l)$ and the secret vector s_L

1: **for** $i = 0$ to L **do**
2: $m[i] \leftarrow ((w - \langle v, s_i \rangle) \bmod q\ ) \bmod 2$
3: **end for**
4: **return** Plain text m

If it is implemented using the school book multiplication method, then it will take more time. In our paper, the main aim is to reduce the time for all these operations. So that it can be implemented on real-time data in a cloud computing environment. We have used the number theoretic transfer method for polynomial multiplication. The concept of NTT has been explained in subsection II-C4 and the NTT algorithm and inverse NTT algorithm are given in algorithms 6 and 7.

Algorithm 6 Algorithm for Number Theoretic Transform

Input: n, x

Output: Output Vector y

Initialization: An input vector x of length n which contains nonnegative integers

1: Working Modulus $M = max(x) + 1$
2: Select k, such that $N = kn+1$ and N is prime and $N \geq M$
3: Calculate the prime factor of $N - 1$
4: Calculate the generator g
5: Such that $a^{N-1/primefactor} \cong 1(\mod N)$
6: Calculate $\omega \equiv g^k(\mod N)$
7: Calculate the output vector y
8: **for** $i = 0$ to $n - 1$ do **do**
9: **for** $j = 0$ to $n - 1$ **do**
10: $y[i] = \sum x[j]\omega^{ij}(\mod N)$
11: **end for**
12: **end for**
13: **return** Output Vector y

Algorithm 7 Algorithm for Inverse Number Theoretic Transform

Input: n, y, M, N, k

Output: Output Vector x

Initialization: An input vector y of length n which contains nonnegative integers. Use the same value of N,M,k which are used in NTT.

1: Let Q= ω^{-1} in $(\mod N)$
2: **for** $i = 0$ to $n - 1$ do **do**
3: **for** $j = 0$ to $n - 1$ **do**
4: $P[i] * n = \sum y[j]Q^{ij}(\mod N)$
5: **end for**
6: **end for**
7: Let d= n^{-1} in $(\mod N)$
8: **for** $i = 0$ to $n - 1$ do **do**
9: $x[i] = P[i] * d(\mod N)$
10: **end for**
11: **return** Output Vector x

IV. SIMULATION, IMPLEMENTATION, RESULTS, AND DISCUSSION

A. DESIGN PARAMETERS FOR SIMULATION OF ALGORITHM

We have proposed hardware simulation of key generation, encryption, and decryption of the BV scheme in our work. Table 2 shows the different parameters for our experimental setup and its size.

TABLE 2. Values of the Parameter.

Parameter	Value	Description
γ	2	Security parameter
n	$2^9 = 512$	Length of secret vector
q	$2^9 = 512 \approx 513$	Odd modulus
m	$m \geq nlogq + 2\gamma = 4612$	A positive integer represents the number of equations
$m1$	8192	Next nearest number 2^{log2m}
A	4612×512	It is a matrix of size $m \times n$, which holds the coefficient of linear equations.
e	4612×1	Error vector or noise vector
r	4612×1	Random binary vector
s	512×1	Secret vector
b	4612×1	Part of a public key. Where $b = (As + 2e)\ mod\ q$

B. MATLAB/WWWWW/PYTHON SIMULATION AND DESIGN FLOW DIAGRAM FOR FPGA

In this section, the FPGA design flow and the pseudo number generation for the public key, secret key, and noise generation requirement of the algorithm as mentioned in sections III-B, III-C, and III-D are explained.

Fig.11 shows the flow diagram for the design of the BV FHE scheme using FPGA. It is divided into different sub-modules. In Table 2 the values of all the parameters for our design specification are given. First, we checked our design entry using MATLAB and Python code. We implemented the key generation using a pseudo-random number generator in VHDL. Further, we simulated the design using the Questasim

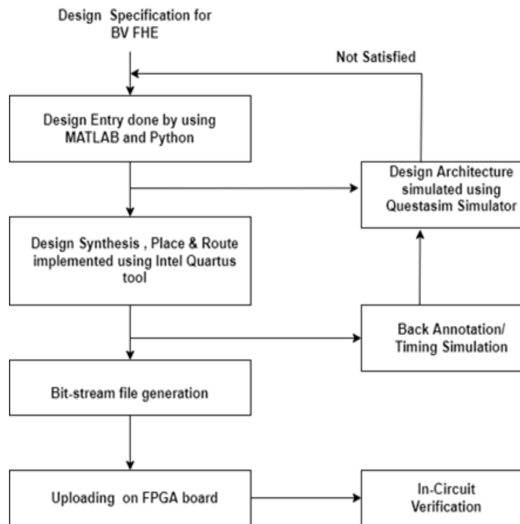

FIGURE 11. Design Flow diagram for FPGA.

simulator. Test vectors generated in MATLAB/Python are used to validate the encryption and decryption operations. Subsequently, the design was implemented using the Quratus place and route tool of Intel. The modular design architecture was implemented considering the logic utilizations, speed, and throughput performance, and design parameters are tabulated in Table 3.

C. ENCRYPTION, DECRYPTION, AND EVALUATION RESULT

Initially, we implemented and tested our method using MATLAB and Python code. The results are shown here. Here the degree of the polynomial is 512(2^9) and the ciphertext modulus is 512(2^9). Two plain texts are PT1=120 and PT2=150 The corresponding ciphertexts are:

Ciphertext CT1(120) is:

v: [158 430 224 316 465 433 279 58 384 157 329 254 393 56 47 65 394 292 429 11 504 368 163 22 146 370 368 208 165 52 415 203 60 109 85 64 4 441 178 288 308 309 168 203 362 363 96 235 365 410 323 304 18 332 386 110 323 140 382 175 161 250 251 43 436 437 371 334 314 85 5 229 272 410 163 441 93 484 204 196 143 242 122 258 372 253 327 225 212 241 58 31 203 421 483 306 353 259 116 340 490 52 161 256 61 507 104 65 289 30 224 330 491 180 139 255 146 177 203 333 143 497 461 333 290 475 436 49 323 125 0 101 98 95 324 59 488 331 316 284 379 471 0 277 5 257 116 442 469 133 285 14 433 505 32 12 343 418 187 506 137 349 203 9 192 74 461 478 217 392 349 100 253 453 432 257 66 8 200 95 338 126 217 296 146 440 280 181 95 455 217 349 500 493 93 507 318 147 163 208 397 27 493 413 430 487 207 305 446 118 266 262 348 406 228 357 480 145 246 452 90 507 345 230 486 11 263 151 103 505 250 158 161 345 325 159 324 343 173 503 344 511 237 151 75 168 22 370 116 415 190 366 115 480 426 272 158 159 343 499 390 79 206 69 97 497 339 63 142 154 313 347 406 492 382 387 280 312 311 397 352 264 389 468 211 499 184 141 245 259 159 182 293 104 444 146 81 133 246 476 207 251 126 508 89 422 17 482 250 78 109 47 251 258 424 54 95 312 211 362 323 114 207 41 101 141 489 246 46 466 193 464 152 342 383 370 110 42 426 444 357 252 410 303 44 68 248 155 220 3 181 384 495 268 322 132 471 38 8 159 307 244 401 95 84 308 450 216 36 482 446 107 62 167 499 235 214 276 225 412 432 153 395 177 314 3 92 382 248 319 209 486 214 0 388 147 47 68 381 112 396 168 105 139 78 21 422 312 98 493 361 427 82 140 51 381 473 261 82 152 242 207 159 275 301 141 142 38 462 262 215 422 385 281 219 466 343 390 325 302 207 165 225 134 209 198 309 293 311 238 209 236 36 397 177 488 505 200 189 209 297 367 250 349 361 138 496 309 356 83 186 431 451 279 473 225 366 152 326 453 473 113 371 391 291 17 262 299 60 406 376 190 252 216 94 247 59 15 260 135 105 334 166 291 37 383 129 265 117 25 76 138]

w: [61 173 226 205 382 129 235 245 489 54 61 106 152 462 445 373 429 358 112 369 257 364 430 491 34 384 221 111 354 442 481 255 139 284 105 112 409 134 72 212 258 339 474 305 139 355 383 159 326 250 102 47 207 15 82 5 278 48 2 472 472 439 71 15 420 418 224 434 163 284 73 258 326 154 71 342 38 83 140 64 427 383 90 98 90 173 463 463 372 340 217 91 115 216 470 464 299 274 378 127 206 110 416 411 410 354 510 335 265 420 463 465 491 455 180 349 477 191 231 291 286 473 130 428 237 508 270 123 97 420 47 361 221 490 380 297 511 395 390 309 243 35 161 291 216 259 381 346 113 403 425 503 285 136 122 63 481 154 346 505 276 274 354 275 495 345 105 148 17 18 267 286 49 304 25 327 323 386 189 253 73 252 332 293 239 152 138 10 287 316 9 299 371 147 203 349 363 293 116 411 413 345 497 359 67 336 335 127 278 354 258 418 235 341 428 231 497 498 44 395 482 124 360 260 97 428 273 206 115 293 309 252 261 430 316 331 97 479 473 258 283 266 126 269 99 413 17 182 354 364 312 74 239 453 406 139 384 465 275 236 99 53 416 432 217 306 65 32 493 231 291 211 201 504 164 127 115 150 73 251 107 29 234 292 325 346 420 201 29 288 271 198 69 415 431 360 18 431 98 108 114 162 488 51 208 469 420 388 455 255 488 73 503 140 63 337 331 153 347 234 192 124 39 38 260 265 227 157 114 173 201 293 173 265 31 496 25 418 485 315 506 183 152 329 42 433 505 64 291 91 214 371 314 124 265 398 144 268 407 398 496 117 42 148 34 19 337 142 478 92 134 377 279 309 149 33 278 200 507 227 450 206 341 132 385 77 18 488 407 431 389 205 240 305 186 214 101 221 176 499 172 230 172 495 497 120 121 432 236 346 187 41 122 127 205 68 331 37 220 284 333 329 282 342 491 493 459 471 482 252 498 323 113 496 205 251 28 121 291 305 311 13 266 71 134 296 43 38 349 18 179 387 474 414 280 334 86 137 271 22 364 190 97 248 492 272 138 163 400 284 96 228 373 204 65 426 264 503 46 188 324 482 466 372 467 482 110 128 277 442 134 358 47 40 295 315 62 88 385 308 66 181 73 167 508 382 181 213 487 192 310 487]

Ciphertext CT2(150) is:

v: [168 39 494 412 101 196 289 161 382 490 196 413 338 68 29 455 205 165 467 439 190 35 455 466 285 270 134 279 243 171 452 143 190 368 124 66 288 105 11 183 495 93 224 86 239 510 242 248 504 244 234 225 488 20 275 288 270 142 487 443 484 48 194 343 332 279 25 308 355 339 386 200 418 369 373 313 253 351 244 468 83 142 355 107 129 15 272 372 260 144 447 393 59 145 1 81 135 119 26 429 334

9 383 26 111 447 316 281 493 268 177 50 23 485 227 408 345 409 112 59 308 203 413 65 395 380 222 188 223 444 458 472 220 251 286 225 485 482 164 454 307 92 145 207 117 279 458 227 321 102 278 338 25 353 467 366 211 224 239 254 84 336 25 94 350 425 157 337 235 162 3 68 221 481 241 305 209 174 312 70 312 131 306 324 331 129 155 160 306 462 67 328 252 460 335 417 102 143 355 361 505 290 267 493 389 367 270 352 304 160 70 181 67 200 150 213 294 107 99 156 84 317 46 374 505 454 245 249 185 222 450 75 423 403 303 223 511 440 208 366 260 11 48 122 264 75 362 129 451 176 383 420 197 496 324 155 86 83 460 0 78 493 456 467 349 460 69 118 470 242 393 389 235 495 486 360 380 265 405 437 262 311 357 134 148 47 226 110 195 272 507 62 127 62 129 354 343 66 334 140 297 122 346 491 303 433 282 484 317 419 124 70 188 11 19 47 54 483 493 74 3 434 141 207 35 411 392 403 494 317 464 490 97 22 17 277 471 131 306 500 244 389 125 79 263 17 5 295 111 508 167 198 473 229 64 223 210 501 337 388 37 419 127 506 406 364 30 338 447 406 63 257 450 492 200 318 256 418 326 190 333 269 486 165 398 504 510 369 137 508 186 179 325 288 238 88 209 432 258 302 285 207 108 143 398 93 376 287 176 293 441 231 166 187 213 68 116 453 198 298 482 96 286 315 144 18 202 364 388 359 507 354 76 93 46 235 142 48 329 70 466 283 14 282 129 352 2 328 42 248 190 446 52 198 379 3 348 110 424 317 103 176 118 123 184 12 182 88 258 413 423 404 45 238 200 394 164 58 464 317 478 511 24 167 473 191 431 440 86 373 37 257 59 411 166 165 502 70 141 213 250 333 274 96 473 383 212 250 497 484 408 168]

w: [231 141 395 117 162 302 387 472 333 337 25 15 473 266 446 242 136 90 314 4 18 10 165 254 461 36 344 315 159 72 107 321 37 487 253 267 14 111 354 138 313 437 349 150 485 96 20 87 325 2 40 57 205 397 455 288 398 386 201 494 385 474 250 123 121 482 297 44 510 2 507 452 141 304 77 391 415 231 178 363 488 279 169 133 219 184 229 236 263 213 389 127 104 452 165 427 98 357 117 279 216 351 399 239 132 400 15 467 88 354 189 511 162 247 15 56 167 206 53 263 147 220 163 310 92 134 105 418 75 374 148 415 385 480 145 25 349 231 445 245 487 256 114 484 201 2 231 35 11 65 32 373 336 236 236 311 111 396 280 217 381 357 419 394 13 248 143 60 43 31 384 147 467 395 258 374 299 267 459 172 196 123 120 196 48 397 344 499 101 45 222 21 232 402 364 283 330 69 472 217 417 186 38 395 215 336 471 378 284 40 56 261 357 293 335 460 42 130 274 166 149 231 213 240 138 439 163 424 223 406 147 480 214 46 459 462 120 130 393 322 211 388 407 127 244 77 394 453 106 281 178 454 208 461 499 42 473 237 276 226 287 435 52 197 265 137 375 390 448 258 61 499 404 139 445 423 355 44 123 240 171 286 321 135 442 57 414 31 253 436 266 29 232 155 258 375 360 481 316 164 407 227 98 389 249 401 73 509 472 138 282 466 494 279 420 229 287 321 104 352 488 222 6 304 208 393 76 212 66 24 511 49 109 324 386 473 157 265 83 44 59 277 315 2711 253 145 94 298 52 262 489 275 284 370 417 162 105 89 263 222 45 210 62 476 287 223 261 62 66 322 184 357 276 66 245 246 331 319 418 349 132 443 96 211 102 11 15 44 428 248 264 83 116 121 279 344 488 411 413 221 289 223 325 89 164 138 498 387 323 504 215 444 75 237 98 361 169 146 206 451 365 236 368 21 87 71

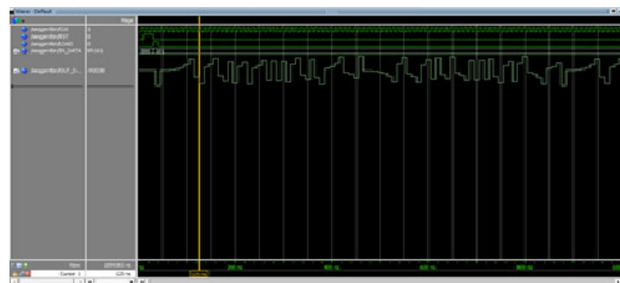

FIGURE 12. Pseudo-Random Number Generator (VHDL Simulator-Questasim).

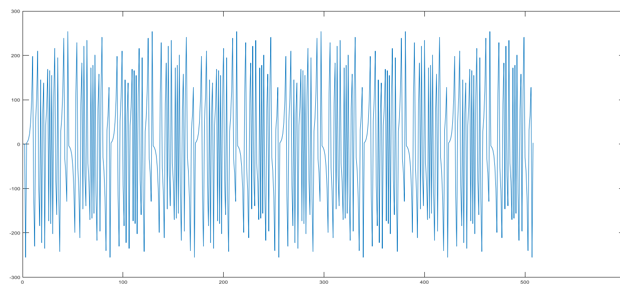

FIGURE 13. Pseudo-Random Number Generator for 512 bits (MATLAB)).

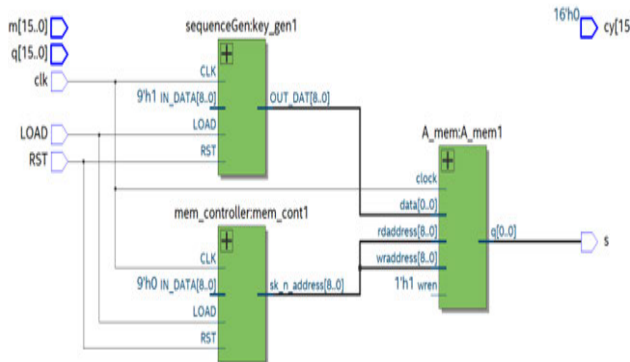

FIGURE 14. Secret Key Generation (VHDL Simulator-Questasim).

184 203 481 255 262 13 234 418 416 454 105 488 413 384 311 477 160 270 99 181 420 48 266 349 383 153 471 429 375 314 231 355 468 400 361 369 198 361 274 368 443 255 35 137 420 406 469 222 151 80 154 446 288 49 307 484 335 446 46 360 402 140 173 149 258 385 50 89 365 354 5 86 40 303 167 420 447 35 121 497 330 94 69 477 302]

Decrypted Result: $CT3(CT1 + 4) : 124$

Decrypted Result: $CT4(CT2 * 20) : 252$

Decrypted Result: $CT5(CT1 + 4 + 20 * CT2) : 120$

D. SIMULATION RESULTS

Fig. 12 shows the simulation result of the Pseudo-Random Number Generator for 512 bits using VHDL Simulator-Questasim.

Fig. 13 shows the simulation result of the Pseudo-Random Number Generator for 512 bits using MATLAB. The correctness of PRNG is validated by comparing the MATLAB and the VHDL simulator data.

Fig. 14 shows the RTL view of the PRNG block implementation to show the actual circuit can be simulated and implemented. The secret key is generated and stored in

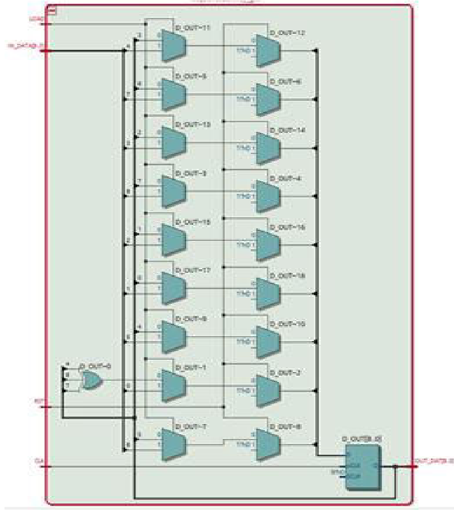

FIGURE 15. Secret Key Address Counter (VHDL Simulator-Questasim).

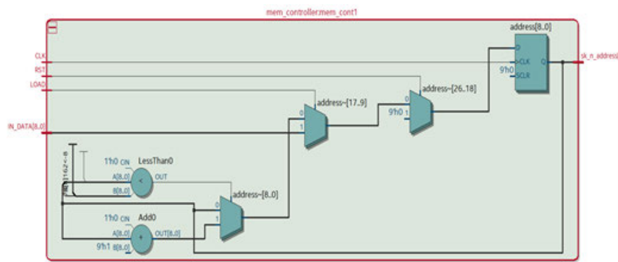

FIGURE 16. Secret Key Memory Controller (VHDL Simulator-Questasim).

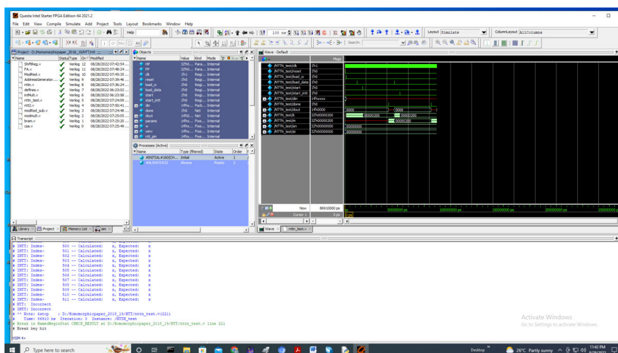

FIGURE 17. NTT Simulation (VHDL Simulator-Questasim).

memory which is block RAM of FPGA. The Fig. 15 and Fig. 16 shows the RTL view of the Pseudo-Random Number Generator and Secret Key Memory Controller.

Based on the above key generation block we have generated the secret key, public key, and random noise block of the algorithm as per Fig. 4.

The simulation of the polynomial multiplier using NTT and INTT in VHDL and Questasim simulation is validated with MATLAB results. The NTT output result is depicted in Fig. 17.

The entire FHE design as per the block diagram is implemented in intel Agilex FPGA and the logic utilization and speed performance have been tabulated. The performance bottleneck is the NTT block. Therefore, we have made a

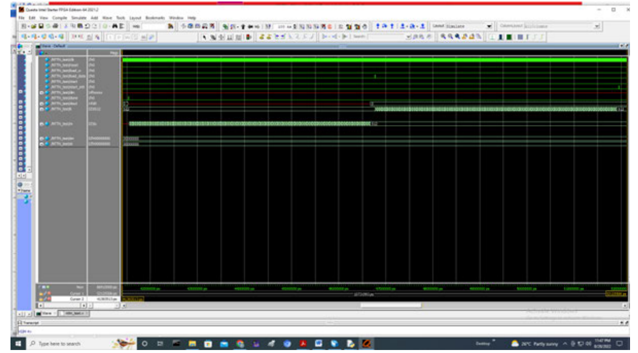

FIGURE 18. Waveform Window (VHDL Simulator-Questasim).

TABLE 3. Synthesis Result for Core NTT.

BF	LUT (ALM s)	No. of dedicated Registers	Block Memory (in bits)	RAM Blocks	DSP Blocks	F_{max}
1	720/ 487,200 (< 1%)	1159	466,944/ 145,612,800 (< 1%)	24/ 7110 (< 1%)	3/ 4,510 (< 1%)	259 MHZ
2	933/ 487,200 (< 1%)	1730	466,944/ 145,612,800 (< 1%)	24/ 7110 (< 1%)	6/ 4,510 (< 1%)	259 MHZ
4	1,724/ 487,200 (< 1%)	2893	466,944/ 145,612,800 (< 1%)	24/ 7110 (< 1%)	12/ 4,510 (< 1%)	254 MHZ
8	3,836/ 487,200 (< 1%)	5344	467,968/ 145,612,800 (< 1%)	34/ 7110 (< 1%)	24/ 4,510 (< 1%)	254 MHZ
16	10,773/ 487,200 (2%)	10472	469,504/ 145,612,800 (< 1%)	52/ 7110 (< 1%)	48/ 4,510 (< 1%)	240 MHZ
32	34,451/ 487,200 (7%)	20758	473,088/ 145,612,800 (< 1%)	106/ 7110 (1%)	96/ 4,510 (2%)	226 MHZ

generic architecture which can be implemented with the various number of butter-fly which is the key processing element (PE). If the numbers of butter-fly are more, the speed of execution is better but the hardware resources consumed by the circuit are more. Further, the speed of the algorithm depends on the maximum clock frequency achievable in the device family. Towards the above requirement, we have carried out the implementation of NTT and logic utilization and frequency (Fmax) achievable by the device. Table 3 shows the implementation results of the NTT algorithm in FPGA.

The polynomial multiplier has been implemented with NTT schemes as per Fig. 10 and the numbers of cycles and times of execution are tabulated in Table 4.

E. TIME REQUIRED FOR KEY GENERATION, ENCRYPTION, AND DECRYPTION

According to Fig. 4 and Table 2, the number of polynomial multiplications with the corresponding dimensions for key

TABLE 4. Performance analysis for NTT-based Polynomial Multiplication.

n	BF	No. of Cycles	F_{max}	Time (μs)
512	1	2304	259	8.895752896
512	2	1152	259	4.447876448
512	4	576	254	2.267716535
512	8	288	254	1.133858268
512	16	144	240	0.6
512	32	72	226	0.318584071
1024	1	5120	259	19.76833977
1024	2	2560	259	9.884169884
1024	4	1280	254	5.039370079
1024	8	640	254	2.519685039
1024	16	320	240	1.333333333
1024	32	160	226	0.707964602
2048	1	11264	259	43.49034749
2048	2	5632	259	21.74517375
2048	4	2816	254	11.08661417
2048	8	1408	254	5.543307087
2048	16	704	240	2.933333333
2048	32	352	226	1.557522124
4096	1	24576	259	94.88803089
4096	2	12288	259	47.44401544
4096	4	6144	254	24.18897638
4096	8	3072	254	12.09448819
4096	16	1536	240	6.4
4096	32	768	226	3.398230088

TABLE 5. Time Taken for different operations without NTT and with NTT.

	Key Generation	Encryption	Decryption	Total Time for BV FHE
Operation	$\mathbf{As}$	$\mathbf{A^T r}, \mathbf{b^T r}$	$\mathbf{vs}$	
Number of Polynomial Multipliers with dimensions	$mx(n*n)$	$nx(m*m),(m*m)$	$n*n$	
Time	5.34 Sec	48.18 Sec, 0.1 Sec	0.001 Sec	53.53 Sec
Number of Polynomial Multiplications using NTT	mx ((3n*log2n) + 2n)	nx (3m1*log2m1+ 2m1), (3m1*log2m1+ 2m1)	3n*log2n+ 2n	
Time	0.33 Sec	0.85 Sec, 0.001 Sec	72 Microsec	1.18 Sec

generation, encryption, and decryption is given in Table 5. Here * represents convolution and x denotes multiplication, m1 is the next nearest value which is $2^{13} = 8192$. So, for the computation of time, we have taken 32 butter-fly-based NTTs. As shown in Table 5, the total time required for key generation, encryption, and decryption without NTT for m=4612 and n=512 is 53.53 seconds. But when it is implemented using 32 butter-fly-based NTTs, the total time required for m=8192 and n=512 is 1.18 second. So, the NTT based system is almost 45 times faster than the hardware system which is implemented without NTT.

F. DISCUSSIONS

In our investigation, it was found that lattice-based FHE algorithms are mathematically similar and the major computation is the polynomial multiplier. Due to the advent of modern FPGAs such as agilex FPGA (10 nanometer) with large numbers of DSP multipliers. The F(max) of 250 MHz is archived for encryption and decryption algorithms. By using large numbers of butter-fly, the long-length plain text may be encrypted and operation of the encrypted text can be carried out. A large number of butterflies is recommended for Cloud Computing which demands huge data encryption and faster speed of execution. Similarly, another hand our architecture with a smaller number of butter-fly can be tailored for IoT applications that demand less data for encryption and decryption and it can be done in moderate time. For, IoT kind of data a lesser number of butter-fly and even smaller FPGAs can be used. For example, for 512 lengths of a polynomial using 1 butter-fly and 32 butter-fly takes 8.8us and 0.6us respectively. The throughput performance of 15x with 32 parallel butter-fly compared to a single butter-fly. So, it is found that 32 butter-fly-based solutions can be used for homomorphic encryption operations in cloud computing.

V. CONCLUSION AND FUTURE WORK

We have developed a new hardware architecture for the BV FHE scheme using FPGA. In this work, we have carried out an extensive simulation of the proposed design in MATLAB, and Python to validate the encryption and decryption algorithm. Further, we have simulated the design architecture of the BV scheme in the Questasim simulator and implemented it in an Intel FPGA using the Quartus tool. In this research work, we found that the proposed architecture indeed takes less time to execute and consumes less FPGA hardware resources. In addition, we have designed a very generic architecture and it can accommodate the change in length of the polynomial for efficient execution time and to use optimal hardware resources available in FPGA. Though we have implemented it in Intel Agilex FPGA, the same design can be extended to any FPGA family.

REFERENCES

[1] Z. Brakerski and V. Vaikuntanathan, "Efficient fully homomorphic encryption from (standard) LWE," *SIAM J. Comput.*, vol. 43, no. 2, pp. 831–871, 2014.

[2] O. Regev, "The learning with errors problem," *Invited Surv. CCC*, vol. 7, no. 30, p. 11, 2010.

[3] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," *J. ACM (JACM)*, vol. 60, no. 6, pp. 1–35, 2013.

[4] C. Gentry, A. Sahai, and B. Waters, "Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based," in *Proc. Annu. Cryptol. Conf.* Cham, Switzerland: Springer, 2013, pp. 75–92.

[5] Z. Brakerski and V. Vaikuntanathan, "Lattice-based FHE as secure as PKE," in *Proc. 5th Conf. Innov. Theor. Comput. Sci.*, Jan. 2014, pp. 1–12.

[6] C. Gentry, "A fully homomorphic encryption scheme," Ph.D. thesis, Dept. Comput. Sci., Stanford Univ., Stanford, CA, USA, 2009. [Online]. Available: crypto.stanford.edu/craig

[7] N. P. Smart and F. Vercauteren, "Fully homomorphic encryption with relatively small key and ciphertext sizes," in *Public Key Cryptography—PKC 2010*. Berlin, Germany: Springer, 2010, pp. 420–443.

[8] J. H. Cheon, J.-S. Coron, J. Kim, M. S. Lee, T. Lepoint, M. Tibouchi, and A. Yun, "Batch fully homomorphic encryption over the integers," in *Proc. Annu. Int. Conf. Theory Appl. Cryptograph. Techn.*, Cham, Switzerland: Springer, 2013, pp. 315–335.

[9] M. Van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, "Fully homomorphic encryption over the integers," in *Advances in Cryptology EUROCRYPT 2010*. Berlin, Germany: Springer, 2010, pp. 24–43.

[10] D. Stehle and R. Steinfeld, "Faster fully homomorphic encryption," Centre Adv. Comput., Algorithms Cryptogr., Dept. Comput., Macquarie Univ., Sydney, NSW, Australia, Tech. Rep. 2010/299, 2010. [Online]. Available: https://ia.cr/2010/299

[11] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "Leveled) fully homomorphic encryption without bootstrapping," *ACM Trans. Comput. Theory*, vol. 6, no. 3, pp. 1–36, 2014.

[12] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," Cryptol. ePrint Arch., Tech. Rep. 2012/144, 2012. [Online]. Available: http://eprint.iacr.org/2012/144

[13] Z. Brakerski and V. Vaikuntanathan, "Fully homomorphic encryption from ring-LWE and security for key dependent messages," in *Proc. Annu. Cryptol. Conf.* Cham, Switzerland: Springer, 2011, pp. 505–524.

[14] O. Regev, *On Lattices, Learning With Errors, Random Linear Codes, and Cryptography*, vol. 56, no. 6. New York, NY, USA: Association for Computing Machinery, Sep. 2009, doi: 10.1145/1568318.1568324.

[15] J.-S. Coron, A. Mandal, D. Naccache, and M. Tibouchi, "Fully homomorphic encryption over the integers with shorter public keys," in *Advances in Cryptology CRYPTO 2011*. Berlin, Germany: Springer, 2011, pp. 487–504.

[16] M. Alkharji, H. Liu, and C. Washington, "Homomorphic encryption algorithms and schemes for secure computations in the cloud," in *Proc. Int. Conf. Secure Comput. Technol.*, 2016, p. 19.

[17] C. Gentry and S. Halevi, "Implementing gentry's fully-homomorphic encryption scheme," in *Proc. Annu. Int. Conf. Theory Appl. Cryptograph. Techn.* Cham, Switzerland: Springer, 2011, pp. 129–148.

[18] C. Gentry and S. Halevi, "Fully homomorphic encryption without squashing using depth-3 arithmetic circuits," in *Proc. IEEE 52nd Annu. Symp. Found. Comput. Sci.*, Oct. 2011, pp. 107–109.

[19] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proc. 41st Annu. ACM Symp. Symp. Theory Comput. (STOC)*, 2009, pp. 169–178.

[20] M. Naehrig, K. Lauter, and V. Vaikuntanathan, "Can homomorphic encryption be practical?" in *Proc. 3rd ACM Workshop Cloud Comput. Secur. Workshop (CCSW)*, 2011, pp. 113–124.

[21] C. Gentry, S. Halevi, and N. P. Smart, "Better bootstrapping in fully homomorphic encryption," in *Proc. Int. Workshop Public Key Cryptogr.* Cham, Switzerland: Springer, 2012, pp. 1–16.

[22] V. Vaikuntanathan, "Computing blindfolded: New developments in fully homomorphic encryption," in *Proc. IEEE 52nd Annu. Symp. Found. Comput. Sci.*, Oct. 2011, pp. 5–16.

[23] M. Tebaa, S. E. Hajji, and A. E. Ghazi, "Homomorphic encryption method applied to cloud computing," in *Proc. Nat. Days Netw. Secur. Syst.*, Apr. 2012, pp. 86–89.

[24] D. J. Wu, "Fully homomorphic encryption: Cryptography's holy grail," *XRDS, Crossroads, ACM Mag. Students*, vol. 21, no. 3, pp. 24–29, Mar. 2015.

[25] Y. Yang, S. Zhang, J. Yang, J. Li, and Z. Li, "Targeted fully homomorphic encryption based on a double decryption algorithm for polynomials," *Tsinghua Sci. Technol.*, vol. 19, no. 5, pp. 478–485, Oct. 2014.

[26] M. Kim, Y. Song, and J. H. Cheon, "Secure searching of biomarkers through hybrid homomorphic encryption scheme," *BMC Med. Genomics*, vol. 10, no. S2, pp. 69–76, Jul. 2017.

[27] X. Yi, M. Kaosar, M. Golam, R. Paulet, and E. Bertino, "Single-database private information retrieval from fully homomorphic encryption," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 5, pp. 1125–1134, May 2013.

[28] S. Behera and J. R. Prathuri, "Application of homomorphic encryption in machine learning," in *Proc. 2nd PhD Colloq. Ethically Driven Innov. Technol. Soc. (PhD EDITS)*, Nov. 2020, pp. 1–2.

[29] C. Gentry, S. Halevi, and N. P. Smart, "Homomorphic evaluation of the AES circuit," in *Proc. Annu. Cryptol. Conf.* Cham, Switzerland: Springer, 2012, pp. 850–867.

[30] X. Cao, C. Moore, M. O'Neill, E. O'Sullivan, and N. Hanley, "Accelerating fully homomorphic encryption over the integers with super-size hardware multiplier and modular reduction," *IACR Cryptol. ePrint Arch.*, vol. 2013, p. 616, 2013.

[31] Y. Su, B. Yang, C. Yang, and L. Tian, "FPGA-based hardware accelerator for leveled ring-LWE fully homomorphic encryption," *IEEE Access*, vol. 8, pp. 168008–168025, 2020.

[32] S. S. Roy, F. Turan, K. Jarvinen, F. Vercauteren, and I. Verbauwhede, "FPGA-based high-performance parallel architecture for homomorphic computing on encrypted data," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2019, pp. 387–398.

[33] E. Oztürk, Y. Doröz, B. Sunar, and E. Savaş, "Accelerating somewhat homomorphic evaluation using FPGAs," *IACR Cryptol. ePrint Arch.*, vol. 2015, p. 294, 2015.

[34] S. Behera and J. R. Prathuri, "FPGA-based design architecture for fast LWE fully homomorphic encryption," in *Cyber Security and Digital Forensics*. Cham, Switzerland: Springer, 2022, pp. 575–584.

[35] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in *Proc. 35th Annu. Symp. Found. Comput. Sci.*, Nov. 1994, pp. 124–134.

[36] R. Agrawal, L. de Castro, G. Yang, C. Juvekar, R. Yazicigil, A. Chandrakasan, V. Vaikuntanathan, and A. Joshi, "FAB: An FPGA-based accelerator for bootstrappable fully homomorphic encryption," 2022, *arXiv:2207.11872*.

SAGARIKA BEHERA (Member, IEEE) received the B.E. degree in computer science and engineering from the Veer Surendra Sai University of Technology (VSSUT), Burla, Odisha, India, in 2001, and the M.Tech. degree in computer science and engineering from VTU, Karnataka, India, in 2009, where she is currently pursuing the Ph.D. degree. She is a Research Scholar with the CMR Institute of Technology, Bengaluru, VTU. Her research interest includes data security in cloud computing.

JHANSI RANI PRATHURI (Member, IEEE) received the Ph.D. degree in computer science from the University of Hyderabad, India. Her research interests include information security and privacy, cryptography, big data, machine learning, cloud computing, and algorithms.