IMPLEMENTATION OF MODULAR EXPONENTIATION USING MONTGOMERY ALGORITHMS

Manish Bansal[1]
Faculty of Technology
UTU,Dehradun,Uttarakhand
manishbansal635@gmail.com

Amit Kumar [2]
Faculty of Technology
UTU,Dehradun,Uttarakhand
amit989762@gmail.com

Aakanksha Devrari[3]
Women Institute of Technology
UTU,Dehradun,Uttarakhand
aks.uit08@gmail.com

Abhinav Bhat[4]
Faculty of Technology
UTU,Dehradun,Uttarakhand
abhinavbhatt91@gmail.com

Abstract—Several algorithms for Public Key Cryptography (PKC), such as RSA, Diffie-Hellman, and Elliptic Curve Cryptography are used for secure communications. These algorithms require modular exponentiation as their basic operation. Modular exponentiation implies repeated modular multiplication which is computationally very costly as the large operands are used. Therefore computation time is very large. This computation time can be reduced by Montgomery multiplication algorithm. Montgomery multiplication algorithm involves three basic phases 1. Conversion of operands from integer domain to Montgomery domain. 2. Multiplication of operands. 3. Conversion of operands back from Montgomery domain to integer domain. A architecture designed to implement Montgomery Modular exponentiation using right to left exponentiation approach, which allows the parallel execution of modular operations "square and multiplications". The implementation of Montgomery modular exponentiation is achieved on Spartan3E, virtex4 and virtex6 series of FPGAs for 4, 8, 16 and 32 bits respectively

Index terms— Cryptography, RSA, Modular multiplication, Montgomery algorithm.

I. INTRODUCTION

Cryptography is the branch of study dealing with information security and authentication techniques. The recent advancements in the wireless communications area and personal communications systems have made providing information security a more and more important subject. This feature becomes more complicated when future technology system requirements and real-time computation speed are considered. To solve these security problems, a lot of research and development is being carried out in this matter, and cryptography has been playing a very important role in any communication system in the recent times.

Cryptography is the science of using mathematics to encrypt and decrypt data. Cryptography enables you to store sensitive information or transmit it across insecure networks (like the internet) so that it cannot be read by anyone except the intended recipient.

Cryptography involves the application of algorithms to transform a message into a representation of the message that is then referred to as the cipher-text. This algorithm must be able to then take that cipher-text and reverse the transformation to obtain the original message. In the field of cryptography, symmetric and asymmetric cryptography constitute two of the major categories of algorithms. Symmetric cryptography is defined by encryption and decryption with a single identical key and is often much more efficient than the alternative method of asymmetric cryptography.

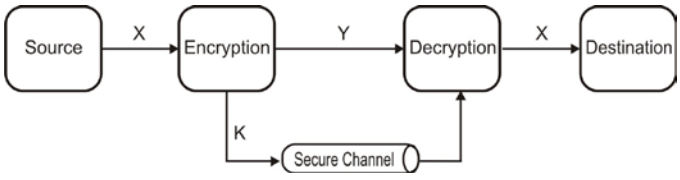

Figure 1 Symmetric Cryptosystem

Asymmetric cryptography has the characteristic of using two different keys in which one key is used for encryption and one key is used for decryption. This allows one or more parties to encrypt messages with a public key, and only the party that possesses the private key to decrypt the messages. Asymmetric cryptography enables digital signatures and public-key infrastructures, but is generally accepted to be much more computationally difficult. Although there are methods to greatly improve the efficiency of certain types of asymmetric algorithms, there is still a large focus to increase the computational efficiency of asymmetric cryptography.

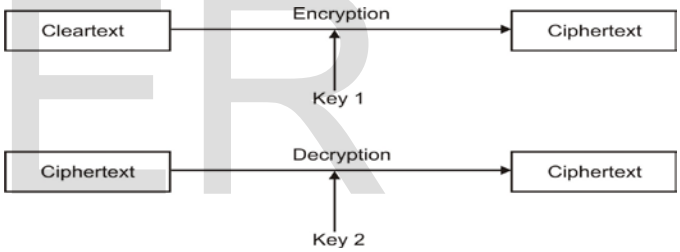

Figure 2 Asymmetric Cryptosystem

II. TYPES OF CRYPTOGRAPHY

There are two kinds of cryptography in this world that is, The one who stops your dear one to read your files and other to stop major governments to read your files.

Cryptography can be ***strong* or *weak***. Simply the weak cryptography is the one which can be decrypt easily by anyone. Cryptographic strength is measured in the time and resources it would require to recover the plaintext. The result of strong cryptography is cipher text that is very difficult to decipher without possession of the appropriate decoding tool. Given all of today's computing power and available time even a billion computers doing a billion checks a second, it is not possible to decipher the result of strong cryptography before the end of the universe. One would think, then, that strong cryptography would hold up rather well against even an extremely determined cryptanalyst. No one has proven that the strongest encryption obtainable today will hold up under tomorrow's computing power. Vigilance and conservatism will protect you better, however, than claims of impenetrability.

III. CRYPTOGRAPHIC TECHNIQUES

The concept of public-key cryptography evolved from an attempt to attack two of the most difficult problems associated with symmetric encryption[1]. The first problem is that of key distribution key distribution under symmetric encryption requires either (1) that

two communicants already share a key, which somehow has been distributed to them; or (2) the use of a key distribution center. Whitfield Diffie, one of the discoverers of public-key encryption reasoned that this second requirement negated the very essence of cryptography: the ability to maintain total secrecy over your own communication.

Asymmetric algorithms rely on one key for encryption and a different but related key for decryption. These algorithms have the following important characteristics:

• It is computationally infeasible to determine the decryption key given only knowledge of the cryptographic algorithm and the encryption key.

In addition, some algorithms, such as RSA, also exhibit the following characteristic.

• Either of the two related keys can be used for encryption, with the other used for decryption.

A scheme has five ingredients:

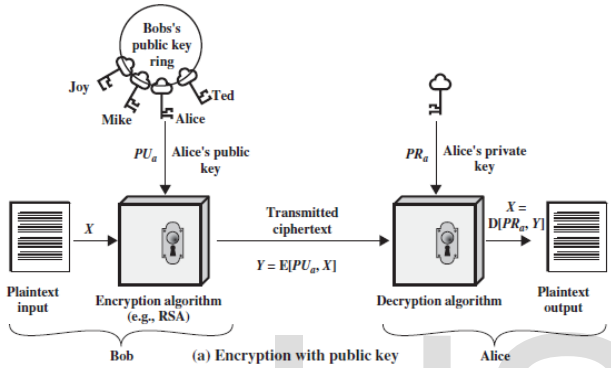

Figure **3** Public Key Cryptosystem [14]

1. **Plaintext:** This is the readable message or data that is fed into the algorithm as input.
2. **Encryption algorithm:** The encryption algorithm performs various transformations on the plaintext.
3. **Public and private keys:** This is a pair of keys that have been selected so that if one is used for encryption, the other is used for description. The exact transformations performed by the algorithm depend on the public or private key that is provided as input.
4. **Cipher text:** This is the scrambled message produced as output. It depends on the plaintext and the key. For a given message, two different keys will product two different cipher texts.
5. **Decryption algorithm:** This algorithm accepts the cipher text and the matching key and produces the original plaintext.

The essential steps are the following:

1. Each user generates a pair of keys to be used for the encryption and decryption of messages.

2. Each user places one of the two keys in a public register or other accessible file. This is the public key. The companion key is kept private As Figure 2.1 suggests, each user maintains a collection of public keys obtained from others.

3. If Bob wishes to send a confidential message to Alice, Bob encrypts the message using Alice's public key.

4. When Alice receives the message, she decrypts it using her private key. No other recipient can decrypt the message because only Alice knows Alice's private key.

With this approach, all participants have access to public keys, and private keys are generated locally by each participant and therefore need never be distributed. As long as a user's private key remains protected and secret, incoming communication is secure. At any time, a system can change its private key and publish the companion public key to replace its old public key.

IV. THE RSA ALGORITHM

The pioneering paper by Diffie and Hellman introduced a new approach to cryptography and, in effect, challenged cryptologists to come up with a cryptographic algorithm that met the requirements for public-key systems. The **RSA** scheme is a block cipher in which the plaintext and cipher text are integers between 0 and n - 1 for some n. A typical size for n is 1024 bits, or 309 decimal digits. That is, n is less than 2^{1024}.

RSA makes use of an expression with exponentials. Plaintext is encrypted in blocks, with each block having a binary value less than some number n. That is, the block size must be less than or equal to $\log_2(n) + 1$; in practice, the block size is I bits, where $2i < n \leq 2i+1$. Encryption and decryption are of the following form, for some plaintext block M and cipher text block C. Both sender and receiver must know the value of n. The sender knows the value of e, and only the receiver knows the value of d. Thus, this is a public-key encryption algorithm with a public key of $PU = \{e, n\}$ and a private key of $PR = \{d, n\}$. For this algorithm to be satisfactory for public-key encryption, the following requirements must be met.

1. It is possible to find values of e, d, n such that $M^{ed} \bmod n = M$ for all $M < n$.

2. It is relatively easy to calculate $M^e \bmod n$ and $C^d \bmod n$ for all values of $M < n$.

3. It is infeasible to determine d given e and n.

For now, we focus on the first requirement and consider the other questions later. We need to find a relationship of the form

$$M^{ed} \bmod n = M \qquad 1$$

The RSA algorithm obtained is illustrated in given figure with an appropriate example also.

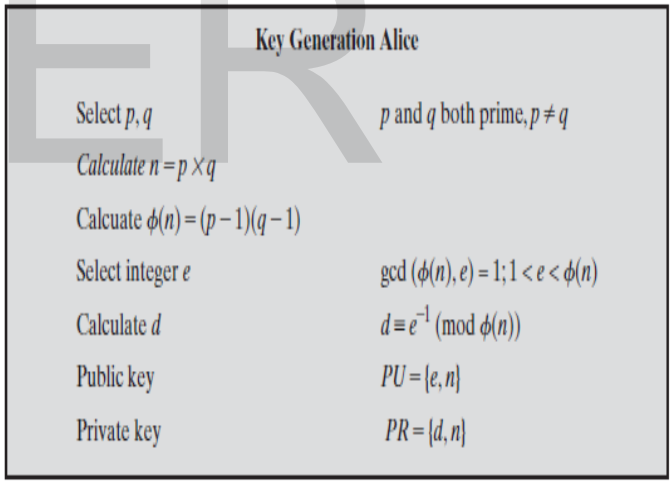

Figure 4 RSA algorithm

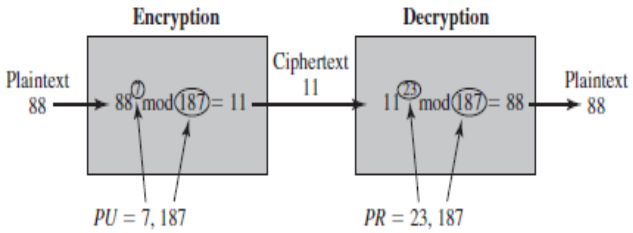

Figure 5 RSA example

V. MONTGOMERY MULTIPLICATION ALGORITHM

Montgomery multiplication is a technique that combines multiplication and reduction into a single operation. It achieves this

by converting values into images within the Montgomery domain, computing the product of those images, and then converting back from the Montgomery domain. An image in the Montgomery domain is defined as x' = x·R mod m. The basic required operation in Montgomery Multiplication is that of Montgomery Product.

Montgomery modular multiplication used to calculate the R-L binary exponentiation is fast and suitable for hardware implementation[4]. It replaces the division by the modulus with simple right shifts, easy to implement but requiring some pre and post calculations.

Inputs: $A = a_{n+1}\, a_n \ldots a_0$, $B = b_{n+1}\, b_n \ldots b_0$, $N = n\ n\text{-}1\ n_n\ \text{-}2 \ldots n_0$,

With $A, B < 2N$, $a_{n+1} = b_{n+1} = 0$.

Output: $S_{n+2} = A \times B \times 2^{-n-2} \bmod N$

Begin

$S_0 = 0$

For i from 0 to n+1 do

1. $q_i = (S_i + a_i \times B) \bmod 2$

2. $S_{i+1} = (S_i + a_i \times B + q_i \times N) / 2$

End For.

End.

Figure 6 Montgomery Multiplication Algorithm

The application of Montgomery algorithm on two numbers A and B of n bits gives the following result:

$$\mathbf{C = MM\ (A, B) = A \times B \times R^{-1} \bmod N} \qquad \mathbf{2}$$

Where $R = 2^n$, and n is an integer with $2^n - 1 < N < 2^n$, such as gcd (R, N) = 1.
Since $R = 2^n$, it suffices to the modulus N to be an odd integer, where this condition is satisfied for the RSA.
The Montgomery algorithm used is without final subtraction represented by algorithm, which allows reduction of the execution time in plus of the occupied area. The architecture of the Montgomery modular multiplication without final subtraction is represented as:

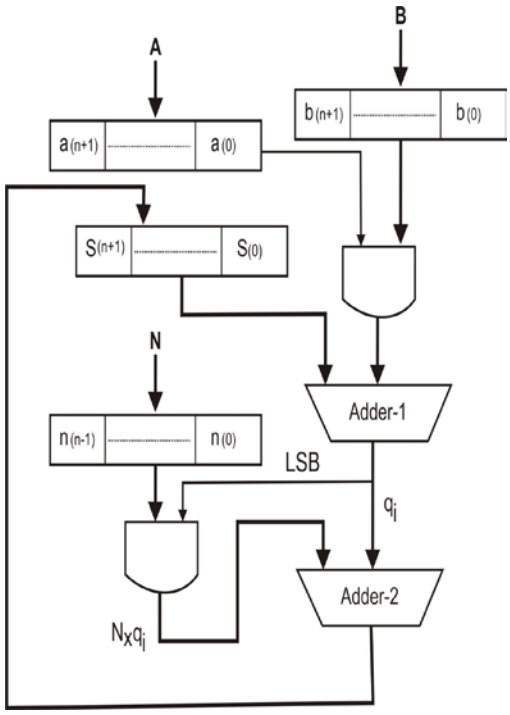

Figure 7 Architecture of Montgomery Multiplication

VI. Modular Exponentiation Architecture

Modular exponentiation is a primary operation in RSA public-key cryptography. There are many different algorithms that are known to improve the efficiency of the modular exponentiation with varying degrees of complexity and each addressing different areas of modular exponentiation, but the basic mathematical operation is:

$$\mathbf{C = M^e \bmod n}$$

Modular exponentiation, realized by a series of modular multiplications, is very costly in computation time for large operands.

Modular exponentiation, realized by a series of modular multiplications, is very costly in computation time for large operands. The simplest and easy method to compute 1024 bits modular exponentiation is the binary method , known as the "Square and multiply"[5]. It is based on scanning the bits of the binary exponent, then a squaring is performed at each step and depending on the scanned bit value, a subsequent multiplication is performed.

To increase the computation performances of this operation:

1. The R-L binary exponentiation is used, which scans the exponent bits from the least significant bit (LSB) and allows the squaring and multiplication to run in parallel.
2. The Montgomery algorithm is used, which replaces the division by the modulus by a series of addition and shift operations in order to reduce the execution time of the modular multiplication[8][9][10].

The Montgomery modular exponentiation algorithm is explained below based on Montgomery multiplication approach:

Inputs: e , M , N, R^2 ;e = $(e_n\text{-}1e_n\text{-}2...\ e_2e_1e_0)_2$
Output: C= M^emod N;
C = Montg(1, R^2, N) , S = Montg (M , R^2 , N);
Begin
For i = 0 to n - 1 do
Begin
a. If (ei = 1) then C = Montg(C, S, N); [multiply]
b. S = Montg (S, S, N); [square]
End;
C = Montg (C, 1, N).
End.

Figure 8 Right-to-Left Exponentiation Algorithm

To adapt the Montgomery multiplication algorithm to the R-L binary modular exponentiation, we must make a mapping in order to enter in the Montgomery domain, where a series of modular multiplications are performed, then a re-mapping is required to recover the true result in the last iteration of the modular exponentiation out of the Montgomery domain [5]. The proposed architecture for fast computing the binary modular exponentiation based on Montgomery multiplication is:

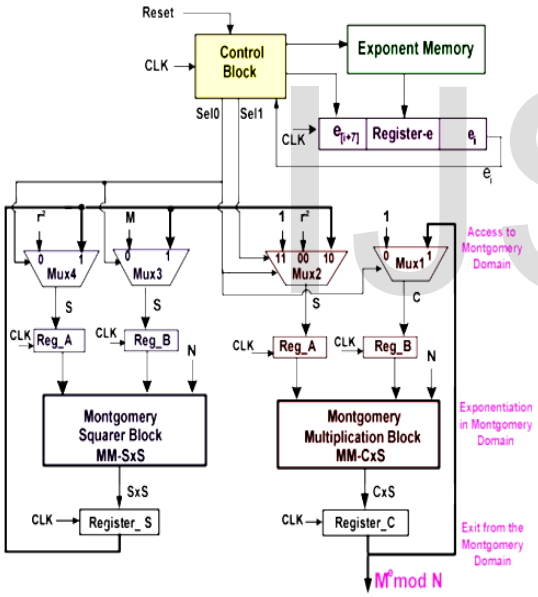

Figure 9 Modular Exponentiation Architecture

It consists of :

1. A control block to manage squaring and multiplication operations. It has as inputs the Reset and the clock CLK.

2. A memory to store the exponent e connected to a shift register for shifting the exponent bits.

3. Two Montgomery modular multiplication blocks MM-SS for squares and MM-CS for multiplications with two registers S and C at their outputs to store respectively the intermediate results of squares and multiplications [6].

4. Two registers A and B at the input of each multiplication block to store the inputs data (M, R^2) or the intermediate results.

5. Two multiplexers at the input of each multiplication block to select the inputs data (M, R^2,1) or the intermediate results.

Entering in the Montgomery domain needs to perform two Montgomery modular multiplications in parallel:

MM (M, R^2 (mod N)) and MM (1, R^2 (mod N)), which gives as outputs: M × R (mod N) and 1 × R (mod N).

- In the Montgomery domain, multiplications and squares are performed in parallel by two Montgomery multipliers and successively until the exhaustion of all the exponent bits [6].
- The multiplexers select the inputs (M × R mod N) and (1 × R mod N) or the outputs of the multipliers according to the logic control to give the result (M^e × R mod N).
- Exiting from the Montgomery domain means performing a last Montgomery multiplication in order to eliminate the r factor from the result to finally obtain (M^e mod N).

VII. CONFIGURING FPGAS

FPGAs can be configured in two ways: in the first case, hardware description languages (HDL) are used to describe the behavior of the circuit and then this description is converted to the gate level net list. FPGA is programmed with that net list. This illustrated below in Figure.

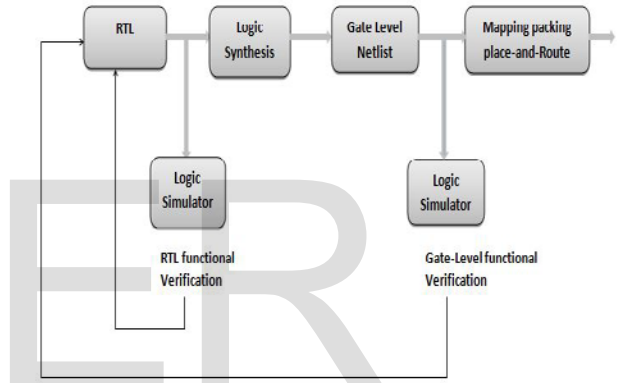

Figure 10 Process flow for configuring FPGAs

In the second case the desired schematic is designed and then converted to gate level net list and FPGA is programmed with that net list.

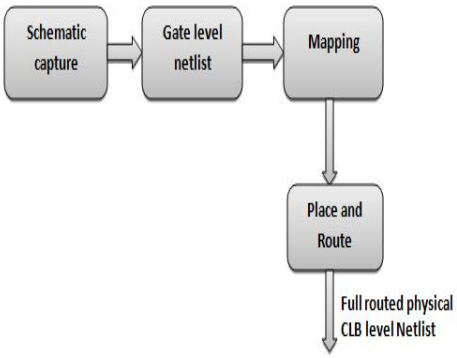

Figure 11 Process flow for configuring FPGAs

As designs grew in size and complexity, schematic based design flows ran out of stream. Visualizing, capturing, debugging, understanding and maintaining a design at the gate level of abstraction became increasingly difficult inefficient and time

consuming for large designs. Thus designers preferred following HDL based design flow.

VIII. SIMULATION TOOL

Initially, to start with the Verilog or VHDL code for a particular design is written and tested. Simulation is done using Mentor's Modelsim for both VHDL and Verilog or other Verilog simulators. Modelsim is a simulation and a debugging tool for VHDL, Verilog, and other mixed-language designs from Mentor Graphics.

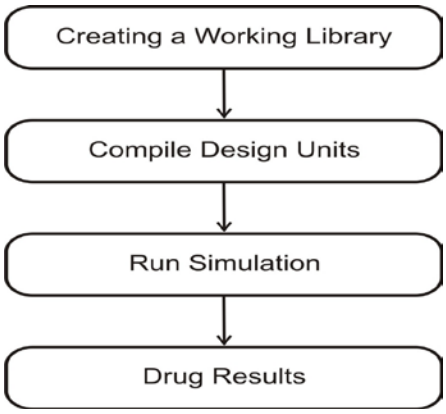

Figure 12 Modelsim simulation flow

To start with a working library is created and the code is compiled using the commands depending upon whether the code is VHDL or Verilog. Verilog Compiled Simulator (VCS) from Synopsys is a ultra-high-capacity, high-performance by the Verilog simulator that incorporate the advance high-level process of abstraction and verification onto an open platform.

The basic work flow for VCS consists of two basic steps:
a) Compiling source files into executable binary files
b) Running the executable binary file

This two-step approach simulates the design faster and uses less memory than other interpretive simulators.

The design is started by writing the Verilog code, which is saved in a file with the extension '.v'. Then the synthesis phase comes, the first step in the synthesis process is compilation. Compilation is the conversion of the high-level Verilog language, which describes the circuit at the RTL level, into a net list at the gate level.

The second step is optimization, which is performed on the gate-level net list for speed or for area. At this stage, the design can be simulated. Finally, the physical layout of the FPGA chip is generated by means of place-and-route software and then FPGA is configured by a programming hardware.

IX. SIMULATIONS RESULTS

The results are based upon following equation:

$$\mathbf{Out = a*b*R^{-1} \bmod m} \qquad \mathbf{3}$$

Where R^{-1} is taken as multiplicative inverse of R in mod m satisfying given equation

$$\mathbf{R*R^{-1} \equiv 1 \bmod m} \qquad \mathbf{4}$$

These are the simulation results obtained in Xilinx ISE (ism simulator) for 16 bits and 32 bits. The results require r_square values as the initialized values.

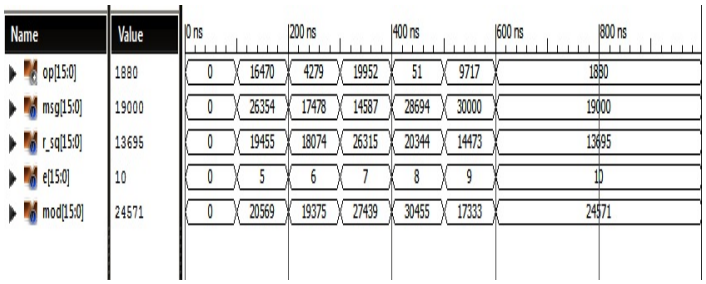

Figure 13 Montgomery modular exponentiation for 16 bits

In this Fig 13 the simulation result of Montgomery modular exponentiation is shown that is of 16 bits. In this various inputs are used that is 'msg' ,'e', 'mod'. and 'r_sq' respectively. Hence the output comes by using these inputs is '1880d'.

The simulated waveform for 32 bit input values (implemented on vertex 6) is given as :

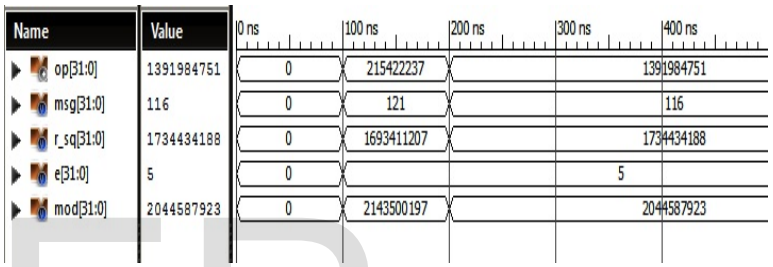

Figure 14 Montgomery modular exponentiation for 32 bits

In this Fig 14 the simulation result of Montgomery modular exponentiation is shown that is of 32 bits. In this various inputs are used that is 'msg' ,'e', 'mod'. and 'r_sq' respectively. Hence the output comes by using these inputs is '1391984751d'.

X. DEVICE UTILIZATION

The synthesis analysis and results are carried out using different FPGA boards according to their capability. The device utilization reports of various FPGA boards (Spartan 3E, virtex4 and virtex6) are given as :

1. Device utilization on virtex 6(xc6vcx130t-2ff484) for 16 bits Montgomery modular exponentiation i**s :**

Table 1 Utilization Summary for virtex 6 for 16 bits

Slice logic utilization	**Used**	**Available**	**Utilization**
Number of slice registers	0	93,120	0%
Number of slice LUTs	33,222	46,560	71%
Number of occupied slices	10,603	11,640	91%
Number of bonded IOBs	79	240	32%

2. Device utilization on virtex 6(xc6vlx550t-2ff1759) for 32 bits Montgomery modular exponentiation is :

Table 2 Utilization Summary for virtex 6 for 32 bits

Logic utilization	Used	Available	Utilization
Number of slice LUTs	319631	343680	93%
Number of fully used LUT-FF pairs	0	319631	0%
Number of bonded IOBs	159	840	18%

XI. CONCLUSION AND FUTURE WORK

A Montgomery Modular Exponentiation algorithm is specified, analyzed, synthesized and implemented using hardware description language Verilog with the help of Xilinx ISE 14.3. This thesis has described the architecture for computing the R-L binary modular exponentiation and its implementation on FPGA. The binary modular exponentiation is computed by a series of squaring and multiplications [15]. Montgomery modular multiplication has been chosen to realize the two operations, which has greatly reduced the computation time of the modular exponentiation.

The architecture allows the parallel execution of squaring and multiplications. The proposed architecture can support key sizes up to 1024 bits using the same circuits and making some changes in the control block with increasing the memory which is still available on the FPGA as only 3% were used. The resources offered by FPGAs such as memory blocks and Carry Save Adders (CSA) have been used advantageously to speed-up the execution time of the modular exponentiation.

As mentioned earlier the architecture presented in this thesis is an early prototype and there are some issues left that need to be addressed to improve performance:

1) Making the word size and therefore the multiplier width independent from the multiplier depth to improve the throughput.
2) Deferring carry propagation until the very end of the algorithm by deploying carry save addition as much as possible.
3) Reducing the start-up latency of the initialization step to improve the pipeline effectiveness.

The Ultimate goal of our research is to create a flexible hardware solution capable of addressing the current and future needs for security of information. As advances are being made in the field of cryptanalysis, the security parameters of cryptographic systems, such as key sizes, but also complete algorithms, need to adapt. To protect investment in the infrastructure of information systems, scalable and flexible architectures become increasingly important. The presented research is one of the first steps towards this goal, but many more must follow.

XII . REFERENCES

[1] Muhammad I. Ibrahim, Mammon B.I. Reaz, Khandaker Asaduzzamanand Sassed Husain, "Hardware Prototyping of an Efficient Encryption Engine", *International Journal of Electrical, Computer and Systems Engineering* 4:4 2010.

[2] N. Neath L.M Morella, "Parallel computation of modular exponentiation for fast cryptography", *Int. Journal of. High Performance Systems Architecture*, Vol. 1, No. 1, Copyright © 2007 Inderscience Enterprises Ltd ,2007.

[3] ErsinO ksuzoglu and ErkaySavas."Parametric, Secure and Compact Implementation of RSA on FPGA", *International Conference on Reconfigurable Computing and FPGAs*, pp.391-396 Dec.2008.

[4] Ming-Der Shieh,Jun-Hong Chen, Hao-Hsuan Wu, and Wen-Ching Lin, "A New Modular Exponentiation Architecture for Efficient Design of RSA Cryptosystem", *IEEE Trans. On VLS I Systems*, Vol 16 N° 9 Sept. 2008.

[5] Colin D. Walter, "Right-to-Left or Left-to-Right Exponentiation"*First International Workshop on Constructive Side-Channel Analysis and Secure Design COSADE* pp. 40-46, 2010.

[6] Xia Hong, Hu Wenhao, Yan Jiangyu, "Design and Implementation of High-Performance Modular Exponentiation Arithmetic Unit", *First International Conference on Information Science and Engineering (ICISE)*, 2009.

[7] "Virtex-4 FPGA User Guide", Xilinx .Dec. 2008.

[8] N.Nedjah L.M Mourelle,"High-performance SoC-based implementation of modular exponentiation using evolutionary addition chains for efficient cryptography", *Applied soft computing , Elsevier journal*, 4302–4311 (2011).

[9] P. L. Montgomery, "Modular Multiplication without Trial Division", Math. Computation, Vol. 44, pp. 519-521, 1985.

[10] Paul Barrett. Implementing the Rivest, Shamir and Adleman Public-Key Encryption Algorithm on a Standard Digital Signal Processor. Lecture Notes in Computer Science: *Advances in Cryptology - Crypto* 86. 263:311-323, 1987.