International Journal of Computer Networks and Communications Security

VOL. 1, NO. 3, AUGUST 2013, 102–109
Available online at: www.ijcncs.org
ISSN 2308-9830

Cryptosystem an Implementation of RSA Using Verilog

Rehan Shams[1], Fozia Hanif Khan[2] and Mohammad Umair[3]

[123]Sir Syed University of Engineering and Technology, Deparment of Telecommunication

[2]Department of Mathematics, Department of Electronics

ABSTRACT

In this paper, we present a new structure to develop 64-bit RSA encryption engine on FPGA that can be used as a standard device in the secured communication system. The RSA algorithm has three parts i.e. key generation, encryption and decryption. The algorithm also requires random prime numbers so a primality tester is also design to meet the needs of the algorithm. We use right-to-left-binary method for the exponent calculation. This reduces the number of cycles enhancing the performance of the system and reducing the area usage of the FPGA. These blocks are coded in Verilog and are synthesized and simulated in Xilinx 13.2 design suit.

Keywords: *RSA, Verilog, Cryptosystem, Decryption, Encryption, Implementation, Key Generation, Modular Exponentiation.*

1 INTRODUCTION

There has been a lot of work going on in the field of cryptography and in the recent years it has increased exponentially. As the usage of communication system increases so does the need for securing data over those channels. Many algorithms are designed to meet these needs. Cryptographic algorithms have two major types: symmetric and Asymmetric [1].

Symmetric cryptography requires sharing of a single key at both ends. The problem is the selection of the key privately. In asymmetric (Public key) cryptography this problem is overcome by using an algorithm that deals with two keys. One key is for encryption and the other one is to decrypt the same message. The idea of publishing one key (the public key) and keeping the other one secrete (the private key) can surely make the whole procedure more secure and protected. Only those will be able to read the massage who may also have the private key as well, it is necessary to have both keys if someone encrypt the massage [2]. RSA algorithm belongs to this type of cryptography. This problem is discussed in many ways [12] has provided the high speed RSA implementation of FPGA platforms, [13] showed the high speed RSA implementation of a public key block cipher-MQQ for FPGA platforms. [14] also provided the implementation of RSA algorithm on FPGA. In this

paper much work has done by the [14] but here we are modifying the our proposed engine for 64 bits RSA encryption. Here we are extended the work given by [14], also other algorithms like LFSR, Miller Rabin, Extended Euclidean and Modular exponentiation have been successfully implemented by using the proposed technique of XILINX ISE 13.2.

As far as the significance of the RSA is concern it can be used as a tool for exchanging the secrete information such as massages and conversation by generating the keys and producing digital signatures. However, the complexity comes from calculating the prime factors of large numbers. This work implements the modular exponentiation operation by simple right-to-left-binary method, which helps to reduce the processing time.

2 OVERVIEW OF RSA

It was developed by Rivest, Shamir &Adleman of MIT in 1977. This method is supposed to be the best and commonly used as a public-key scheme which is based on exponentiation in a finite (Galois) field over integers modulo a prime. The security is due to cost of factoring large numbers. As already explained in RSA cryptosystem there

are two key, the public and private key. The public key is advertised to the world and the private key is supposed to kept secret. Therefore an anonymous person will not be able do decrypt the encrypted message if he does not have the private key. The safety depends upon the length of the key, longer the key-length much safer is the data [3].

Following are the steps involved in the RSA algorithm:

2.1 Key generation

Key generation is the most important aspect of RSA Algorithm.

The steps are as follows:

- "Select two random prime numbers p and q
- Calculate n = p x q
- Calculate ø(n) = (p – 1) x (q – 1)
- Select integer e such that gcd (ø(n),e) = 1;1<e<ø(n); where e & ø(n) are relatively prime
- Calculate d = e-1 mod ø(n)"

According to the procedure the encryption key e is available but the decryption key d is not known to all. Mathematically this procedure is defined as, M is the actual message, C is the converted message or cipher text by using publicly available encryption key e, and d is the decryption key.

$$C = M^e \pmod m$$
$$M = C^d \pmod m$$

RSA encryption and decryption are mutual inverses and commutative [4].

3 RSA ALGORITHM

RSA algorithm is divided into blocks and each block is then implemented.

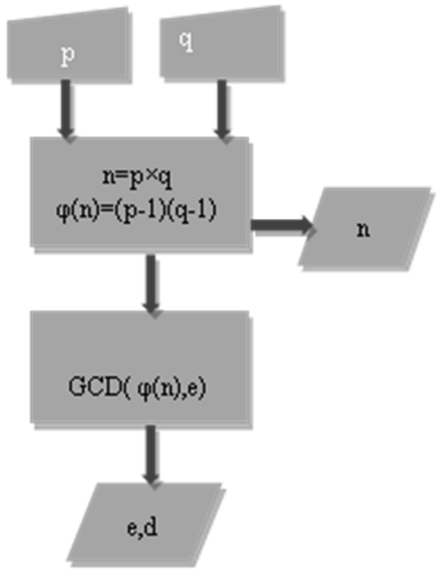

Fig. 1. RSA key generation

The first step is the generation of public and private keys which is summarized in fig.1. It starts with a pseudorandom number generator that generates 32-bit pseudo numbers. These pseudorandom numbers are stored in a FIFO. The pseudorandom number generator will stop working as the FIFO is full. Random numbers from the FIFO are pulled out by the primality tester as this happens the PRNG will start again to make sure that the FIFO remains filled. Coming back to the primality tester, it takes a random number as input and check for the number to beprime. If the number is proved as prime, it goes to the Prime FIFO. The primality tester only pulls a number out of the FIFO when the prime FIFO is not full. When there is a requirement of new keys, two random prime numbers are extracted from the prime FIFO. The structure is shown in Fig. 2.

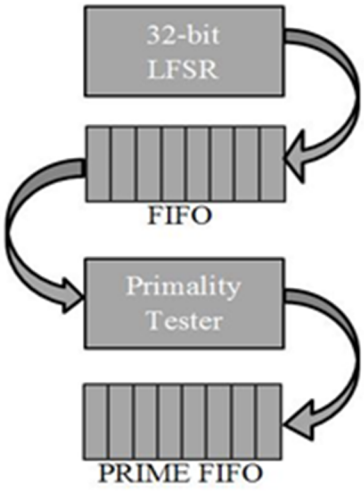

Fig. 2. Prime Random Number Generation

These two numbers are used to calculate n and Φ(n). Φ(n) is forwarded to the Greatest Common Divider (GCD) calculator where a number e is selected which will be the public or encryption key if it satisfies the condition that GCD (Φ(n), e) = 1. This will prove that modular inverse d of this number exists and the modular multiplicative inverse will be our private our decryption key. We got e, d and n, for encryption and decryption. Modular exponentiation is applied to encrypt or decrypt the data. This is something that has to be focused because the performance of RSA algorithm depends on how modular arithmetic functions are calculated. They are the core of the algorithm. This process is shown in Fig. 3:

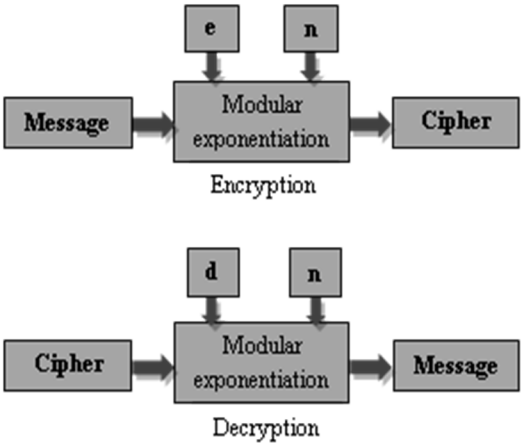

Fig. 3. Encryption and Decryption

3.1 Random Number Generator

The Linear Feedback Shift Register (LFSR) is among the most useful techniques used for generating pseudorandom numbers. Here 32-bit pseudorandom numbers are generated using LFSR. LFSR generates a periodic sequence means that the pattern in which numbers are generated will be repeated after certain interval.When using a primitive polynomial, maximum length of an LFSR sequence is 2n-1.A 32-bit LFSR will produce a sequence of over 4 billion random bits, or 500 million random bytes.The polynomial used for generating this sequence of 32-bit pseudorandom numbers is as under.

$$P(x) = x32+x22 +x2+x1 +1$$

3.2 Primality Tester

The numbers generated by the LFSR consists of both prime and composite numbers. One of the most commonly used primality test is the Miller-Rabin test which is considered as a fastest among all, the main purpose of this test is to separate the prime numbers and then saved them for further use. The Miller-Rabin primality test is one of the fastest and most widely used primality tests that produce good result in polynomial time [5]. We eliminate the even numbers generated from the LFSR and feed the odd numbers to the primality tester. The following pseudo code shows the Miller-Rabin primality test algorithm [6].

"Input: n > 2, an odd integer to be tested for primality;
k, determines the accuracy of the test
Output: composite if n is composite, otherwise probably prime
write n − 1 as 2s·d with d odd by factoring powers of 2 from n − 1
LOOP: repeat k times:
pick a randomly in the range [2, n - 1]
x ← ad mod n
if x = 1 or x = n − 1 then do next LOOP
for r = 1 .. s − 1
x ← x2 mod n
if x = 1 then return composite
if x = n − 1 then do next LOOP
return composite
return probably prime"

The odd integer n from the LFSR to be tested, kdetermines the numbers of round for the test. For each round, we use a new integer 'a', the upper and the lower bound for the integer is 2 and n-2 respectively, which may be 1 or 0 for the composite or probable prime numbers. If the result of any round is 0 (composite), then there is a possibility that n is composite and we will not move any

further. Since there is high probability that n is prime if the result of all the rounds is 1 (probable prime). It's very rare that probable prime n might be composite [7].

3.3 GCD

For the generation of keys, two prime numbers are extracted from the Prime FIFO. Applying respective operations on these two primes gives n and Φ(n). After Φ(n), a number e is selected which obeys the condition GCD (Φ(n),e) = 1, means that e is relatively prime to Φ(n). This will prove that modulo inverse of e exists.

$$e \times d \bmod (\Phi(n)) = 1$$

Extended Euclidian algorithm is used and implemented for this purpose. When the GCD is 1, the module returns the values of e and the modular multiplicative inverse d. Otherwise, e gets an increment of 2 and GCD is calculated again, this is repeated until the value of e satisfies the condition and a positive inverse is found. e will be used as the encryption key and d as the decryption key. The pseudo code for Extended Euclidian algorithm is as under.

```
extended_euclidean_main(p,q)
    e =1
    (gcd, d) = (0, 0)
while (gcd != 1 || d < 0)
begin
        e = e + 2
        (gcd, d)= extended_euclidean_loop((p – 1)(q – 1),
e)
end
return (e,d)

extended_euclidean_loop(a,b)
    (y, y_prev) = (1, 0)
while b != 0
begin
        (y, y_prev) = (y_prev – a/b*y, y)
        (a, b) = (b, a mod b)
end
return (a, y_prev) {gcd( (Φ(n), e) is a, inverse is y_prev}
```

3.4 Encryption/Decryption

Encryption is the process of converting plain text in such a way that eavesdroppers or hackers cannot read it, it is called as cipher text. Decryption is the inverse process by which cipher text is converted back into the form that is readable namely plain text. After generation of the keys, RSA encryption and decryption is done using the mathematical operation $C = M^e \pmod n$ and $M = C^d \pmod n$ respectively. Hence encryption/decryption is just a modular exponentiation operation.It involves few modular operations like modular addition, modular subtraction and modular multiplication.

4 MODULAR EXPONENTIATION OPERATION

Modular Exponentiation operation is simplified using square and multiply algorithm. It is done by using right-to-left-binary method. The purpose of using the binary method is to calculate Me by using the binary expression of exponent e. In binary method the exponentiation operation is broken in to a series of squaring and multiplication. This method is also very useful for speeding up the exponentiation calculation. The LSB binary exponentiation algorithm (also called as right-to-left binary exponentiation algorithm), starting from the least significant bit position it calculates the exponent e and proceeding towards left, which can be write as follows [8].

Input: M, e, n
Output: $C = M^e \bmod n$
Let e contain k bits
If $e_{k-1}=1$ *then C=M else C=1*
  For i=0 to k-1
  $C=C\times C$
  If $e_i=1$ *then* $C=C\times M$

This algorithm works on the principle of scanning bits from the right For every iteration, i.e., for every exponent bit, the current result is squared, If and only if the currently scanned exponent bit has the value 1, a multiplication of the current result by M is executed following the squaring [9].

5 HARDWARE IMPLEMENTATION

For hardware design and implementation, the RSA Cryptosystem is divided into 4 modules.

i. Initial module
ii. Modular exponentiation
iii. Core algorithm
iv. Top module

6.5 Initial Module

This module consists of a 32-bit LFSR for generating random numbers for the algorithm, which are than stored in the FIFO if they are proven to be odd. As this FIFO fills completely the Miller Rabin primality tester takes a number out from the FIFO and test it for prime. The exponentiation part of this algorithm is done by using the right to left

binary algorithm implemented for encryption/decryption. If the test succeeds the number is stored in PRIME FIFO for later use by the algorithm. This process will only stop when the PRIME FIFO is full.

6.6 Modular Exponentiation

The most important and time consuming part of RSA algorithm is calculating the modular exponentiation of a number. For this purpose we implement the Square and Multiply algorithm by using the right-to-left-binary approach. It speeds up the exponent calculation and limits the number of cycles needed.The exponent function is also required in miller and Rabin tester so this module can be called there for processing and calculating, saving both space and time.

6.7 Core Algorithm

Here we implement the basic functions and steps of RSA algorithm. This is further divided into two steps:

- Key generation
- Cryptography

When a new used comes to the system this module takes two numbers as input. These numbers should be 32-bit prime. n and Φ(n) are calculated by inserting them into the multiplier, hence getting a 64-bit number.Φ(n) is then used to find the encryption and decryption keys. For this purpose the Euclidean Algorithm is used which calculates the GCD of Φ(n) and an odd number. If the GCD is found to be 1 this proves that the modular inverse of the odd number exists and the number is co-prime. This number is then saved in a register and will be used as the encryption key. The Extended Euclidean Algorithm is then used to find the decryption key which is the modular inverse of the encryption key. Hence the key generation process is completed for this user and he is allotted with a public and private key that user will use to encrypt and decrypt the data.

Once the keys are created they can be used for encrypting and decrypting the message. The Modular Exponentiation is used for this purpose. Register e_d switches the exponent value so that it could be used for both encryption and decryption by just changing the exponent value to the respective key.

6.8 Top Module

The top module controls the functions of the other modules and interconnects them so as the RSA Algorithms flow is maintained. This module implements a controller with multiple checks so as to get the desired results.

Word size [bits]	Clock frequency
128	65.532
136	68.31
160	70.763
208	72.546
288	81.325

From the above table we can see that the frequency can be decrease by the proposed encryption engine. There is a falling rate of 15% in each case of word size.

Logic utilization	used	Available	utilization
No. of slice	432	587	73%
Number of slice flip plops	7632	9512	80%
No. of 4 inputs LUTs	568	931	61%
No of bonded IOBs	105	323	32%
No. of BRAMs	8	20	40%
No. of MLT 18X18SIOs	15	20	75%
No. of GCLKs	6	24	25%

- Minimum Period: 12.473ns
- Maximum frequency 58 MHz
- Maximum combinatorial Path delay 10ns
- Maximum output required time after clock 8.657

6 SIMULATION RESULTS

LFSR, Miller Rabin, Extended Euclidean and modular exponentiation have been successfully written and tested on Xilinx ISE 13.2. The simulation shows the desired results of these algorithms. The following section shows the simulation results of these algorithms.

6.1 Linear Feedback Shift Register

The pseudo random numbersare generated by a 32-bit Linear Feedback Shift Register which is simulated in Xilinx ISE.

Fig. 4. Simulated Waveform for Pseudo-Random number generator

Fig. 4 shows some 32-bit random numbers that are generated by the LFSR. Only odd numbers will be saved for further processing.

6.2 Miller Rabin Primality Check

Primality tester is implemented using Miller Rabin Primality Test which is simulated in Xilinx ISE.

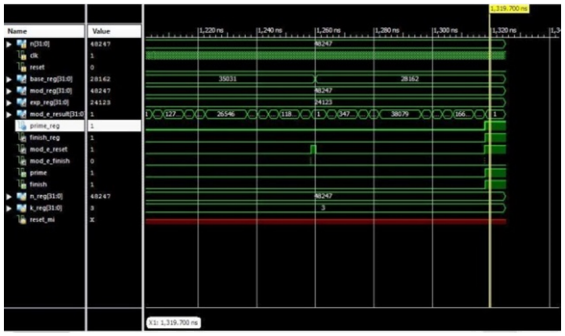

Fig. 5. Simulated Waveform for Primality Tester

Fig.5 shows the simulation of primality test, the input is 2750263. The tester checks whether the number is prime or not and after processing returns that result 2750263 as prime by changing the value of prime_reg to 1.

6.3 Extended Euclidean

Fig.6 shows the Extended Euclidean algorithm simulated in Xilinx ISE 13.2.

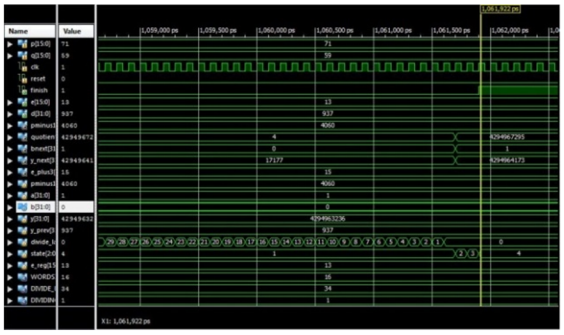

Fig. 6. Simulated Waveform for GCD

The waveform for extended Euclideanalgorithm for the calculation of GCD and modular inverse shows that two inputs are given p=71 and q=59, as a result the algorithm gives e=13 and d=937 and output.

7 MODULAR EXPONENTIATION

Modular exponentiation is used for encryption and decryption. There simulated waveforms are shown in Fig. 7 and Fig.8 respectively.

7.1 Encryption

Fig.7 shows the encryption of a plain text. n is set to 4189 and the exponent is 13 i.e. encryption key. The plain text is 101.

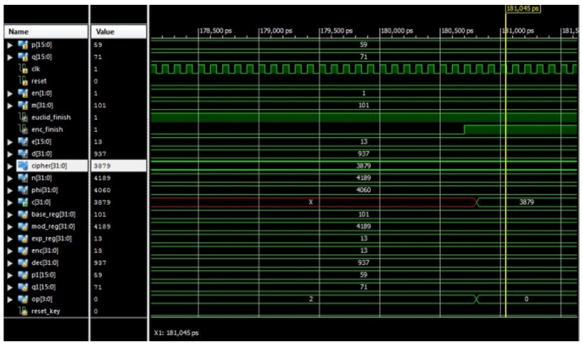

Fig. 7. Simulated Waveform for Encryption

After encryption the resulted cipher text is 3879.

7.2 Decryption

Fig.7 shows the decryption of the cipher text. The value of n remains the same, but for decryption the exponent's value changes to 937 i.e. the value of decryption key. The cipher text is 3879.

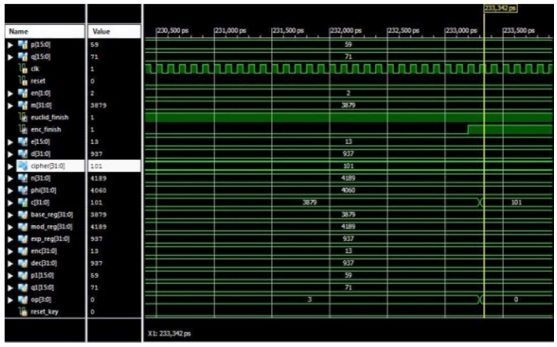

Fig. 8 Simulated Waveform for Decryption

After decryption it converts the cipher text back to the original message i.e. 101.

Table 3: Hardware performances

Algorithm Name	104 bit RSA Encrypt/ decrypt	160-bit MQQ Encrypt/ decrypt	128-bit AES Encrypt/ decrypt	64- bits RSA Encrypt/ decrypt
FPGA type	Virtex-5 XC5VL X30-3	Virtex-5 XC5VL X70T-2	Virtex-5	XILINX 13.2 Verilog
Frequency	251MHz	276.7/24 9.7 MHz	325MHz	254MHz
Throughput	40Kbps	4427Gb ps/399.0 4Mbps	3.78 Gbps	9.8Mbps

From the above table we can see that the results are compared with the other implementations. The implementations are compared using throughputs rate. The results of our implementations shows that in hardware, the proposed method for public key algorithm in encryption and decryption can be made fast as a typical block cipher and is several orders of magnitude faster than other algorithms.

7 CONCLUSION

Here we implemented a 64-bit RSA circuit in Verilog. It is a full-featured and efficient RSA circuit this includes Primality testing, key generation, data encryption and data decryption. We have implemented random number generator using 32-bit LFSR, Miller-Rabin primality test, GCD and modular inverse algorithm using extended Euclidean algorithm and Encryption and Decryption using Modular multiplication and modular exponentiation algorithms (R-L binary algorithms). Each sub-component and top module of RSA was simulated in Xilinx and proved functionally correct. This can easily scale up to large bits such as 512 or 1024 or even longer.

8 ACKNOWLEDGMENT

The authors would like to express sincere gratitude to the Research Centre, Sir Syed University of Engineering and technology for providing fund for the research.

9 REFERENCES

[1] Symeon (Simos) Xenitellis, "A guide to PKIs and Open– source Implementations", The Open–source PKI Book

[2] VibhorGarg, V. Arunachalam ,"Architectural Analysis of RSA Cryptosystem on FPGA", International Journal of Computer Applications (0975 – 8887) Volume 26– No.8, July (2011).

[3] William Stallings, "Cryptography and Network Security Principals and Practices", 4th edition, Pearson Education, Inc., (2006).

[4] Sushanta Kumar Sahu, ManoranjanPradhan, "FPGA Implementation of RSA Encryption System", International Journal of Computer Applications (0975 – 8887), Volume 19– No.9, April (2011)

[5] Monier, L. "Evaluation and Comparison of Two Efficient Probabilistic Primality Testing Algorithms." Theor.Comput.Sci. 12, 97-108, (1980).

[6] Rabin, M. O. "Probabilistic Algorithm for Testing Primality." J. Number Th. 12, 128-138, (1980).

[7] Joe Hurd, "Verification of the Miller-Rabin Probabilistic Primality Test", Computer Laboratory, University of Cambridge

[8] Chia-Long WU, "An Efficient Montgomery Exponentiation Algorithm for Cryptographic Applications", INFORMATICA, Vol. 16, (2005) No. 3, 449–468.

[9] AnkitAnand, Pushkar Praveen, "Implementation of RSA Algorithm on FPGA", International Journal of Engineering Research & Technology (IJERT), Vol. 1 Issue 5, July – (2012).

[10] http://rosettacode.org/wiki/Miller-Rabin_test.

[11] Alkhatib, Mohammad. "On The Design of Projective Binary Edwards Elliptic Curves Over GF (P) Benefiting From Mapping Elliptic Curves Computations to Variable Degree of Parallel Design", International Journal on Computer Science & Engineering/09753397, 20110401.

[12] Thomas wokinger, “High Speed RSA Implementation of FPGA Platforms”, MS thesis Institution of Applied Information Processing and Communications, Graz University of Technology, (2005).

[13] Mohammad El- Hahidy, Danilo G. and Sevin J. K., “High performance Implementation of public key block cipher-MQQ, for FPGA Platforms”,(2004), eprint.iacr.org/2008/339.pdf .

[14] Ankit A. Pushkar P., “Implementation of RSA Algorithm on FPGA”, International Journal of Engineering Research and Technology, Vol. 1, Issue 5, pp. 1-7, (2012).