FPGA Implementation of RSA Encryption Algorithm for E-Passport Application

Khaled Shehata, Hanady Hussien, Sara Yehia

Abstract—Securing the data stored on E-passport is a very important issue. RSA encryption algorithm is suitable for such application with low data size. In this paper the design and implementation of 1024 bit-key RSA encryption and decryption module on an FPGA is presented. The module is verified through comparing the result with that obtained from MATLAB tools. The design runs at a frequency of 36.3 MHz on Virtex35 Xilinx FPGA. The key size is designed to be 1024-bit to achieve high security for the passport information. The whole design is achieved through VHDL design entry which makes it a portable design and can be directed to any hardware platform.

Keywords—RSA, VHDL, FPGA, modular multiplication, modular exponential.

I. INTRODUCTION

E-PASSPORT is a passport that includes a smart card embedded in the back. This card contains the traveler’s personal data. Many countries adopted e-passports to facilitate people traveling and Visa issuing. About 53 different countries including the United State (US) and Canada have used e-passport [1]. However, the security and integrity of the e-passport are very critical. The International Civil Aviation Organization (ICAO) created sets of e-passport standard [2], [3]. 1024-bit RSA is one of the recommended algorithms used for Active Authentication (AA) protocol. This protocol is used to prevent e-passport cloning [4].

The RSA is a public key encryption algorithm invented by Rivest, Shamir, and Adleman in 1977. RSA operation is based on modular exponentiation which requires repeated modular multiplications. Moreover for security reasons RSA operand sizes is recommended to be 1024 bits or more [5]. As a result the modular operations for 1024 bits or higher make RSA is difficult to achieve a high throughput. To address this problem many algorithms are invented such as add and shift, Montgomery multiplication and carry save adder (CSA) [6], [7].

This paper presents the implementation of RSA encryption/decryption algorithm with 1024-bit key length on FPGA. RSA algorithm adopts square and multiply algorithm for modular exponential. The modular multiplier is implemented using add and shift algorithm presented.

The paper is organized as follows: Section II explains RSA algorithm, Section III explains the mathematical algorithms used to execute RSA algorithm. Then in Section IV is discussed the RSA implementation and shows the simulation results. Finally, Section V draws the conclusion.

II. RSA ALGORITHM

RSA is a public encryption algorithm which has a public key for encryption (e) and private key for decryption (d). RSA algorithm is summarized to three main steps [8].

A. Key Generation

In this step the private and public keys are generated as shown in Fig. 1 by:
1. Choose two large prime numbers p and q.
2. Compute modulus number n = p x q.
3. Calculate the Euler function φ(n) = (p-1) x (q-1).
4. Select an integer number e randomly as a public key. It should satisfy Greater Common Divisor GCD(e, φ(n)) = 1, 1< e < φ(n).
5. Compute the private key d such that d x e =1(mod φ(n)).

B. Encryption

In RSA both plain text (M) and cipher text (C) are blocks with length less than \[\log_2 n\]. In encryption, the cipher text is generated by

\[C = M^e \mod n\] (1)

C. Decryption

The decipher text is recovered using the private key (d) by

\[M = C^d \mod n\] (2)

Fig. 1 Block diagram of RSA encryption and decryption algorithms

III. RSA MATHMATICAL OPERATIONS

The RSA encryption/decryption algorithm is based on computation of modular exponentiation operation. The strength of RSA depends on the difficulty of factoring the modulus n to get the prime numbers p and q. Hence, the larger
prime numbers the harder the factorization of modulus n. Therefore the modular exponentiation operation becomes harder to accomplish on a hardware platform. This section details the main modular mathematical operations used for hardware implementation.

a) Modular Exponentiation Operation

Modular exponentiation for large numbers is considerably difficult to compute. Therefore, this operation can be simplified into series of modular multiplication and squaring operations [9], [10]. This algorithm is known as square and multiply algorithm. In this algorithm the exponent number e is scanned either from Left to Right (LR) or Right to Left (RL). In LR method, which is common used, if the scanned bit is logic zero a squared operation is performed. However if the scanned bit is logic zero a squared operation is performed. However if the scanned bit is logic one a multiplication operation is computed. This operation is performed k-time where k is the modulus length. The square and multiply algorithm is described by the following code [8]-[10].

Input:  m, e and  n.
Output: c = m^e mod n,  e > 1
Initialization  c = m if e[k-1] = 1 else c=1
for j = k-1 downto 0 do
  c = c * c  mod n
  if ( e[j] == 1) then
    c = c * m mod n
  end for
return c

b) Modular Multiplication Operation.

The modular multiplication operation is essential to compute the exponentiation modular as shown in previous algorithm. Shift and add algorithm is one of the algorithms used to perform modular multiplication. This algorithm computes y x z (mod n). The numbers y and z are k-bit integers and yi and zi are the ith bit of y and z respectively. The detailed algorithm is described as follows [10], [11].

Input:  y, z, n
Output: Mul = y x z mod n
Initialization Mul = 0;
For i = 0 to k
  Mul=Mul +(y×zi)
if (Mul<0) then
  Mul = Mul / 2;
else
  Mul=(Mul + n) /2;
return Mul;

IV. SIMULATION RESULTS OF RSA ENCRYPTION/ DECRYPTION HARDWARE IMPLEMENTATION

The RSA Encryption / Decryption modules with key length 1024 are designed and implemented based on VHDL code. The design adopts the square and multiply algorithm for modular exponentiation. The modular multiplier is performed based on add and shift algorithm. The public and private keys are generated using C# program. The results are stored in ROM. There are two different ROMs, one is used to store (n, e) keys and the other is to store (n, d) keys. The design is simulated using Xilinx ISE 12.3 targeting Virtex-5 XC5VTX240T-2FF175 FPGA from Xilinx.

A. Add and Shift Algorithm Simulation Results

As discussed before add and shift algorithm is used to perform modular multiplier. It computes Mul = y x z mod n. As shown in Fig. 2, the algorithm inputs are ‘mpand’, ‘mplier’, and ‘modulus’ each with length 1024 bits. These inputs represent y, z and n respectively. The algorithm output is ‘product’ signal which represents ‘Mul’ output. As shown in Fig. 1 mpand = e_{hex}, mplier = 3_{hex} modulus = 21_{hex} and product = 9_{hex} (9 = e x 3 mod 21). At clock: 209.048ns

![Fig. 2 Simulation results of add and shift algorithm](image-url)
### B. Square and Multiply Algorithm Simulation Results

#### Similarly the square and multiply algorithm is designed and tested. This algorithm computes \( c = m^e \mod n \). As shown in Fig. 3 the applied inputs are 'indata', 'inexp', 'inmod' and the output delivered is 'cipher'. These signals represent \( m, e, n \) and \( c \) respectively. From the simulation results shown in Fig. 2, indata = 11_{16}, inexp=903ad9_{16}, inmod = 3b2c159_{16} the output cipher = 36cf344_{16} (36cf344 = 11^{903ad9}_{16} \mod 3b2c159). At [6,771.000ns] clock.

#### RSA Encryption/Decryption Algorithm

The whole system is tested by applying 1024-bit plain text. The used public keys are loaded from ROM module. The simulation result is shown in Fig. 4. By applying the generated cipher text on the RSA decryption algorithm the deciphering output is identical with the original plain text as shown in Fig. 5.
The algorithms in Table I are used firstly R. Srinivasan [13]: Square and multiply algorithm is used for modular exponent and the Montgomery algorithm is used for modular multiply. R. Srinivasan uses 54.7 MHz clock frequency.

Secondly, Mostafizur Rahman [14]: Square and multiply algorithm is used for modular exponent and the add and shift algorithm is used for modular multiply. Mostafizur Rahman uses 100 MHz clock frequency.

Thirdly, Sushanta Kumar Sahu [12]: Square and multiply algorithm is used for modular exponent and the Montgomery algorithm is used for modular multiplier. Sushanta Kumar Sahu uses 101.06 MHz clock frequency.

It is clear that when the number of bits is increased; the frequency is decreased, compared with other works. The previous work is much more complicated than the present work; because of the length of RSA public key and private key under1024 bit are insecurity. There are certain procedures in the selection of the p,q and e in addition to the generation of security code compared to previous work in the latest papers.

In this proposed technique, the Xilinx’s virtex5 xc5vtx240t-2ff175s is used, number of slice (LUTS) used is 28,350, while the available (LUTS) is 149,760 so the proposed technique utilizes 18%. The key generation is 1024 bit, it is the most security code compared to previous work in the latest papers.

V. CONCLUSION

In this paper, a detailed implementation technique for 1024-bit RSA encryption/decryption algorithm is presented. The modular exponential for encryption and decryption process is performed by using square and multiply algorithm. The add and shift algorithm is used to perform the modular multiplier.

All these algorithms are implemented using VHDL code targeting Virtex-5 XC5VTX240T-2FF175 FPGA from Xilinx. The whole design is tested using Xilinx ISE 12.3 tool. The system speed achieved is 36.3 MHz which comply with the speed of smart card used in e-passport.

REFERENCES