A Parallel Method for RSA Cryptosystem Utilizing Topological Architecture

Masumeh Damrudi1*, Kamal Jadidy Aval1 and Norafida Ithnin2

1Department of Computer Science, Firoozkooh Branch, Islamic Azad University, Firoozkooh, Iran; m.damrudi@gmail.com
2Department of Computer System and Communication, Faculty of Computing, University Teknology Malaysia, Skudai - 81310, Johor Bahru, Malaysia

Abstract

Users of today’s digital applications are increasing dramatically and there has been a rapid growth in the users of embedded wireless systems that usually transfer confidential information. These systems do expect robust, real time, and accurate performance, which bring Quality of Service (QoS) into mind. Parallel computing is one of the solutions to such issues to speed up the response time. On the other hand, cryptography is an important part and the first way of keeping information private. As a cryptography method, asymmetric encryption process needs massive mathematical operations, especially when a greater key is needed. This paper presents a new parallel method for RSA cryptography based on mesh topology named MRSA. This method is applicable in embedded devices as a coprocessor and can act as a crypto-engine. The MRSA method is analyzed mathematically and compared to other methods. The result shows that the proposed method has fewer steps as well as multiplication operations to compute the encrypted value compared to the accepted method, which is the binary method.

Keywords: Cryptography, Mesh, Parallel, RSA, Topology

1. Introduction

Users of the digital world expect a certain behavior from applications and consumers of electronic services are subject to higher demands. Embedded systems use a weak CPU due to their processing needs, and intruders attack utilizing strong processing resources. Using a coprocessor to collaborate with CPU would be a suitable idea for these devices. These coprocessors may gain benefits of parallelism, which is an appropriate way of doing operations in a faster way. Cryptographic algorithms usually need massive operations; in result, speedup could be provided using a coprocessor which is benefiting from parallelism. RSA is an asymmetric cryptographic algorithm which is still being used in variety of applications, although; it has been used many years. Having a strong mathematical background, RSA is still secure using greater key lengths. Asymmetric keys solve the key distribution problem in symmetric cryptography. They are also used for non-repudiation, which helps to prove that only the sender who has the key could have sent the message.

Existing parallel approaches on cryptographic algorithms are summarized in. Focusing on the RSA, there has been some pieces of research on performing it in parallel. Recently interconnection network concept has been employed to provide a faster way of doing RSA cryptography. The TRSA and its optimization are parallel cryptographic approaches based on the RSA and tree interconnection network. As far as we know, there is no other significant topology based cryptographic approach other than TRSA. Regardless of TRSA, the so called approaches have not discussed the time complexity or the order of the algorithms which are inseparable from parallel processing. To the best of the authors’ knowledge, the only existing discussion on time complexity, which is the number of multiplications, are the well-known CRT.
Montgomery\textsuperscript{21}, and the binary\textsuperscript{20} where the latter is an accepted method.

In this paper, we define a new parallel cryptographic approach based on mesh interconnection network, which is a combination of parallel cryptographic algorithm and parallel architecture that is called MRSA. The order of this approach is compared to the existing approaches as well as TRSA in terms of number of multiplication. The rest of this paper is organized as follows. Primitive RSA is described in the second section. Afterwards, MRSA parallel encryption algorithm is presented, formulated and analyzed. Finally, the results are discussed and the conclusion is given.

2. Primitive RSA

The RSA details are mentioned in almost all cryptography literature. As a brief explanation, the RSA assumptions are in the following where \( p \) and \( q \) are two large prime numbers:

\[
\begin{align*}
    n &= pq \quad (1) \\
    \varphi &= (p-1)(q-1) \quad (2) \\
    e &< n, \gcd(e, \varphi) = 1 \quad (3) \\
    d &= e^{-1} \mod \varphi \quad (4)
\end{align*}
\]

Considering \( m \) as plaintext, it should be divided into blocks smaller than \( n \). \( C_i = m^e \mod n \) is the encryption and \( m_i = C_i^d \mod n \) is the decryption operation. The variable \( i \) is used to indicate the block numbers. As mentioned in the introduction, the greater the key is, the more secure the data transfer will be\textsuperscript{48}.

3. MRSA Parallel Encryption Algorithm

Whenever some processing elements come together and collaborate to solve massive problems in a reasonable time, a parallel computing environment is formed. In the MRSA method, mesh topology is used as parallel processing architecture. This architecture has \( \sqrt{N} \) diameter where \( N \) is the number of processor element. Each intermediate node is connected to four other nodes, each side node is connected to three nodes, and each corner node is connected to two nodes.

We have considered 10 nodes in our design which nine of them form the mesh, and one is the coordinator. From another point of view, the proposed architecture can work as a cryptographic coprocessor, which collaborates with the CPU. In this case, the coordinator can be the CPU itself. Digital embedded systems do have a CPU, and this coprocessor is used to perform the cryptographic operations faster and with higher throughput. However, the number of nodes can be more or less but not less than five. Number of nodes should follow a mesh rule which is \( S^2 \) where \( S \) is the length of mesh; in addition, a processor element is added as the coordinator of first operation. Using more nodes, the more parallelization is gained but employment of more processor elements for smaller data and key lengths should be avoided. A tradeoff between the number of processor elements, the data size and the key length exists. We have chosen a mesh with nine nodes to describe our solution.

3.1 Mesh Topology Concept

Mesh is a two-dimensional network, which is obtained by arranging \( N \) processor elements into an \( S \times S \) array, where \( S = \sqrt{N} \). A simple mesh is shown in Figure 1 for \( S = 4 \). Every mesh has \( S^2 \) nodes that are connected to their immediate neighbors. The processor in row \( j \) and column \( k \) is denoted by \( P(j, k) \), where \( 0 \leq j \leq S - 1 \) and \( 0 \leq k \leq S - 1 \). Four communication lines link \( P(j, k) \) to its neighbors \( P(j + 1, k), P(j - 1, k), P(j, k + 1), \) and \( P(j, k - 1) \)\textsuperscript{22}. Processor elements on the boundary rows and columns have less than four neighbors and hence less connections.

Mesh is a fixed-degree network\textsuperscript{23}. The degrees of the interior processor elements, the four corner processor elements, and the remaining edge processor elements of a mesh are 4, 2 and 3 respectively. A mesh has \( S \) rows and \( S \) columns. Therefore, transporting a piece of data from the northwest processor to the southeast processor requires traversing \( S - 1 \) rows and \( S - 1 \) columns. A message originating from one corner of the mesh and traveling to the opposite corner of the mesh(diameter) requires traversing a minimum of \( 2S - 2 \) communication links\textsuperscript{23,24}, which has the order of \( O(\sqrt{N})^{25} \).

3.2 The MRSA Method

The MRSA method is employing mesh topology to carry out RSA encryption in a faster way. The following definitions are used for the algorithm.

\[
ed_d : \sum_{i = -1}^{S - 3} 2^{S+i}
\]
The following definitions are used for the algorithm.

The MRSA method is employing mesh topology to carry out RSA encryption in a faster way.

3.2 The MRSA Method

\[ e_o = e \text{ div } e_d \quad (5) \]
\[ e = e \text{ mod } e_d \quad (6) \]

The scheme of the MRSA using mesh topology is demonstrated in Figure 2(a). It should be considered that there is no need for these processor elements to be full-function processors. They just need to be able to perform multiplication and division operations. They don’t need to be able to perform other operations. Receiving the data, doing the operation and sending the result are the only tasks of these processor elements. There is no need for router, routing algorithm and complex hardware devices.

PEs (Processor Elements) are simple digital circuits as shown in Figure 2(b). The modulo operation will be done based on \( n \). The value of \( n \) will be in place, which means \( n \) will be fed into the PEs as the initial step.

In the first step, a processor element \( p \), calculates \( m^{e_o} \mod n \) and sends it as the input of processor element \( p_0 \) to send it as two inputs of processor elements \( p_x \). Then the result of equation \( m^{e_o} \mod n \), which is calculated in \( p \), is sent to processor element \( p_o \) to send it as input of processor elements \( p_x \). In this scheme, Some PEs even don’t perform multiplication and modulo and just send the input values to the next PE. The outputs of \( p \) are \( A \) and \( B \) which are computed as following.

\[ A = m^{e_x} \mod n \quad (7) \]
\[ B = m^{e_s} \mod n \quad (8) \]

The processor element \( p_0 \) receives \( A \), and sends it to \( p_3 \). Afterwards, \( p_3 \) sends \( B \) to \( p_0 \). The processor element \( p_3 \) uses \( A \) as two inputs and calculates \( A^2 \mod n \) and sends the results to \( p_4 \) and \( p_5 \) as its inputs. The processor element \( p_1 \) doesn’t perform any calculation and sends the received value which is \( B \) to \( p_s \). The processor elements \( p_4 \) and \( p_5 \) compute \( A^4 \mod n \) as their inputs is \( A^2 \mod n \). The processor element \( p_6 \) sends its results to \( p_7 \) but \( p_7 \) sends it to \( p_4 \) and \( p_5 \). It means that one input of \( p_7 \) is from \( p_4 \) and the other one is from \( p_5 \). Just like \( p_3 \), \( p_4 \) and \( p_5 \) doesn’t perform any calculation and sends the received value which is \( B \) to \( p_7 \). The processor element \( p_7 \) perform the computation of \( A^4 \mod n \) with its inputs \( (A^4 \mod n) \) and \( p_2 \) computes \( A^{4\times B} \mod n \). The processor elements \( p_0 \) and \( p_8 \) send their values to the \( p_7 \). The output of \( p_8 \) is the encryption of \( m \). The pseudo code of MRSA algorithm is illustrated in Figure 3.

3.3 MRSA Generalization

Apart from this simple example of a 3×3 mesh, the mesh can be of any size. The steps of the algorithm and the way processor elements should behave to do the necessary computations is formulated as following:

\[ \forall i, i \in \{ i \geq 0 \ and \ i \leq S-1 \}; p_i \text{ sends its input to output} (9) \]
∀ \forall j, l \in \{j \geq 1 \text{ and } j < S\}, l \in \{l \geq 0 \text{ and } l < S\}; p_{(j \times s) + l} \text{ computes MulMod operation in brief. If } p_{(j \times s) + l} \text{ has one input, the other input value will be assumed to be the same as the received value.}

Based on the Equation (9), the processor elements \(p_1\) and \(p_2\) only send their results to their successor node and don't perform any computation. Therefore, we can have the following formulas for a 3×3 mesh.

\[
\begin{align*}
   p_1 &: p_2 : B \\
   p_3 &: A^2 \mod n \\
   p_4 &: A^4 \mod n \\
   p_5 &: A^8 \mod n \\
   p_6 &: A^{12} \times B \mod n
\end{align*}
\]

In this example, \(S=3\), therefore:

\[
e_d = \sum_{i=1}^{S-1} 2^i = 2^2 + 2^3 = 12
\]

A known fact in number theory is:

\[
(a \times b) \mod n = ((a \mod n) \times (b \mod n)) \mod n
\]

Based on this fact, the following computations are used to prove that the output of the \(p_8\) is the encrypted message. Output of \(p_8\) is the result of \(A^{12} \times B \mod n\). Therefore:

\[
C = (A^{12} \times B) \mod n = ((A^{12} \mod n \times B \mod n) \mod n
\]

\[
= ((A^{12} \mod n \times B \mod n) \mod n
\]

It is derived from Equations (7) and (8) that:

\[
= (\left(\frac{m^e \mod n}{12}\right)^{\frac{m}{e}} \times \left(\frac{m^e \mod n}{12}\right) \mod n)
\]

Utilizing Equation (17):

\[
= \left(\left(\frac{m^e \mod n}{12}\right)^{\frac{m}{e}} \times \left(\frac{m^e \mod n}{12}\right) \mod n\right)
\]

And using Equations (5) and (6):

\[
C = m^e \mod n
\]

The ciphertext resulting from MRSA is the same as original RSA.

4. Performance and Analysis

The MRSA method has improved the modular exponentiation process as following to do faster computations:

\[
r = e \mod e_d
\]

\[
m^e = m^{e_1} \times \ldots \times m^{e_d} \mod n
\]

\[
m^e = m^{e_1} \prod_{i=1}^{e_d} m^{e_i}
\]

In the MRSA algorithm only \(p\) should be enough powerful to compute modular exponentiation, and the other ones should do at most one MulMod operation. Some processor elements are even simpler and just pass the received value. There is no need to raise \(m\) to the power of \(e\), and \(p\) just needs to raise \(m\) to the power of \(e/e_d\) (in this example \(e/12\)) and this reduction makes the computation faster. Using binary exponentiation in \(p\), we can achieve a more appropriate execution time. The performance of the algorithm is explained briefly as following.

The MRSA is a new method to calculate RSA, which its results are highly depending on the number of MulMod operations. The more blocks of MulMod are used, the more speedup will be gained, and fewer steps will be applied to calculate the total value.

As described in the literature from 8,26–28, the number of multiplications in binary method for the worst case is 2\((k-1)\) and for the best case is \((k-1)\) where \(k\) is the bit length of the exponent, which is \(e\). The MRSA method is mainly divided into two parts. The first part of MRSA is based on the binary method. Let \(x\) be the mesh diameter,
A and B in Equations (7) and (8), as the powers, will be divided to the number of PEs, which is \( \sum_{i=1}^{x-3} 2^{x+i} \). Hence, \( 2 \left( k - 1 - \log \left( \sum_{i=1}^{x-3} 2^{x+i} \right) \right) \) multiplications in the worst case will be applied to compute A and B consequently. Thus, the number of multiplications of the first part in the worst case in the coordinator is:

\[
k' = 2 \left( k - 1 - \log \left( \sum_{i=1}^{x-3} 2^{x+i} \right) \right)
\]  

(20)

The total number of multiplications for the mesh as the second part is \( x^2-x \) since x PEs do not carry out multiplication, thus the total number of multiplications for the best case will be \( 2 \left( k - 1 - \log \left( \sum_{i=1}^{x-3} 2^{x+i} \right) \right) + x^2-x \).

The other subject that should be discussed is the concurrency of the multiplications. Not only the number of multiplications is decreased in this approach, but also there are some multiplications performing at the same time in different PEs, which decreases the time complexity. Although, the total number of PEs carrying out multiplication mesh is \( x^2-x \), where x is the diameter of mesh, the total time for performing the multiplications is \( 2x-1 \), due to the concurrency of PEs. In result, the number of multiplications for the MRSA is \( 2 \left( k - 1 - \log \left( \sum_{i=1}^{x-3} 2^{x+i} \right) \right) + 2x-1 \).

Having the convenient formula for geometric series:

\[
\sum_{k=0}^{n} ar^k = \frac{a(1-r^{n+1})}{1-r}
\]  

(21)

Therefore,

\[
\sum_{i=1}^{x-3} 2^{x+i} = \sum_{i=0}^{x-2} 2^{x+i-1} = \sum_{i=0}^{x-2} 2^{x-1}2^i = \frac{2^{x-1}(1-2^{x-2+1})}{1-2}
\]

\[
\frac{2^{x-1}(1-2^{x-2+1})}{1-2} = \frac{2^{x-1} - 2^{2x-2}}{-1} = 2^{2x-2} - 2^{x-1}
\]

(22)

The reduction of the number of multiplications is reasonable. Using the calculation method from (5), the number of multiplication is illustrated by \( \eta(k,x) \). Eventually, the total number of multiplications in the worst case is:

\[
\eta(k,x) = 2(k - 1 - \log(2^{x-1})) + 2x - 1
\]

\[
\eta(k,x) = 2(k - 1 - \log(2^{x-1}) + \log(2^{x-1} - 1)) + 2x - 1
\]

\[
\eta(k,x) = 2(k - 1 - (x - 1 + \log(2^{x-1} - 1))) + 2x - 1
\]

\[
\eta(k,x) = 2(k - x - \log(2^{x-1} - 1)) + 2x - 1
\]

\[
\eta(k,x) = 2(k - \log(2^{x-1} - 1)) - 1
\]  

(23)

The above Table 1 shows that the number of multiplication operations of primitive RSA will be reduced using the MRSA method compared to the other methods as well as the TRSA. This improvement depends on the number of MulMod blocks in the mesh topology and the length of RSA cryptographic key.

The speedup of MRSA to the binary method in the worst case is calculated using the following formula:

\[
\text{Speedup} = \frac{2(k-1)}{2(k-\log(2^{x-1} - 1))-1} = \frac{k-1}{k-\log(2^{x-1} - 1)-0.5}
\]  

(24)

Figure 4(a) shows the MRSA method using multiple bit key lengths and mesh diameters based on the formula of the worst case. Figure 4(b) presents the same figure from another point of view to figure out the speedup for each configuration.

Table 1. compares the number of multiplications in the average, and worst case for binary, CRT, Montgomery, TRSA(19), and MRSA

<table>
<thead>
<tr>
<th>Method</th>
<th>Average Case</th>
<th>Worst Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary</td>
<td>General form</td>
<td>Criteria</td>
</tr>
<tr>
<td></td>
<td>1.5(k-1)</td>
<td>1535</td>
</tr>
<tr>
<td>TRSA</td>
<td>1.5(k-1)-0.5l</td>
<td>1505</td>
</tr>
<tr>
<td>MRSA</td>
<td>1.5(k-\log(2^{x-1}-1))+0.5x-1</td>
<td>1495</td>
</tr>
<tr>
<td>CRT</td>
<td>3k^2/4+k</td>
<td>787456</td>
</tr>
<tr>
<td>Montgomery</td>
<td>2k^2+k</td>
<td>2098176</td>
</tr>
</tbody>
</table>
Figure 4 (a) shows that exploiting more PEs in MRSA will result in better performance for greater key lengths. The increase of speedup in Figure 4 (b) is depicted with a red arrow.

Selecting the number of processor elements for a mesh is dependent to the area and the speed, which is desired for the security needs of the target system. There is a tradeoff between speed and number of PEs. It must be considered that the number of PEs must be increased to a place that the overhead of multiple PEs does not increase the multiplications which leads to reduction in speed.

5. Conclusion

The MRSA is carrying out RSA cryptography employing parallel mesh topology. Using more secure bit key lengths such as 1024, 2048, and 4096 the MRSA has more suitable speed than the well-known methods. This method uses a coprocessor, which is made up of MulMod blocks that collaborates with CPU. Considering $p$ as the main CPU of an embedded device, the MulMod blocks in mesh topology represent the coprocessor.

The speedup of MRSA method to the previous methods depends on the number of MulMod blocks. As it is seen from the Table 1 and Figure 4, MRSA method is better in terms of the number of multiplications. This improvement is increased using more MulMod blocks. As the number of MulMod operations decreases, the steps as well as the execution time of the MRSA will be decreased.

The architecture of this method is designed to be scalable and can be resized to suit the needed performance. A hardware solution outperforms a software solution as it is almost the case\textsuperscript{29,30}. It's a cheap way of achieving high performance.

6. References

7. Qing L, Yunfei L, Lin H. On the design and implementation of an efficient RSA variant. 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE);