An Efficient Pipelined Technique for Signcryption Algorithms

Ghada F. El Kabbany, Heba K. Aslan and Mohamed M. Rasslan

Informatics Department, Electronics Research Institute
Cairo, Egypt

Abstract

Signcryption algorithms are based on public key cryptography. The main advantage of signcryption algorithms is to provide both confidentiality and authenticity in one step. Hence, signcryption algorithms lower both communication and computation overheads. This reduction, in communication and computation overheads, makes signcryption algorithms more suitable for real-time applications than other algorithms that combine encryption and digital signature in separate blocks. Although, the signcryption algorithms overcome the communication overhead problem, they still suffer from the need to perform arithmetic modular operations. The arithmetic modular operations have high computation overhead. In this paper, we use a pipelined technique to reduce the computation overhead of signcryption algorithms. We apply the proposed pipelined technique to Rasslan *et. al* signcryption algorithm. The proposed pipelined technique is suitable for the selected signcryption algorithm. Rasslan *et. al.* signcryption algorithm is more efficient than all the previously presented algorithms. Rasslan *et. al.* signcryption algorithm allows the recipient to recover the message blocks upon receiving their corresponding signature blocks. This makes Rasslan *et. al* signcryption algorithm perfect for some real-time applications. Also, we illustrate the performance analysis of the proposed solution. The performance analysis shows that the proposed technique reduces the computation time required to execute Rasslan *et. al.* signcryption algorithm with respect to its corresponding values of sequential execution.

Keywords: *Cryptography, Authentication, Encryption, Signcryption, Pipelining, and Modular Multiplication*

1. Introduction

Many real-time applications demand the necessity to lower the communication and computation overheads that is required to provide security. Signcryption algorithms [1-5], which are based on public key cryptography, could be a solution to such applications. Their main advantage is to provide both confidentiality and authenticity in one step. This lowers both communication and computation overheads, which makes signcryption more suitable for real-time applications. The term signcryption was originally introduced and studied by Zheng in [1] with the primary goal of reaching greater efficiency than that can be accomplished when performing the signature and encryption operations separately. Although, the signcryption techniques overcome the communication overhead problem, they still suffer from the need to perform modular operations which requires large computation operations. In the literature, many solutions were proposed to improve the performance of cryptographic algorithms using pipelined techniques [6-14]. A pipeline is composed of a series of producer stages, each one depends on the output of its predecessor. Pipelines are used in cases where a parallel loop cannot be used. With the pipeline pattern, the data is processed in a sequential order, where the first input is transformed into the first output, the second input into the second output, and so on. In a simple pipeline, each stage of the pipeline reads from a dedicated input and writes to a particular output. All the stages of the pipeline can be executed at the same time, because concurrent queues prevent any shared inputs and outputs. Pipelines are useful specifically when the data elements are received from a real-time event stream. In addition, pipelines are used to process elements from a data stream (i.e. in compression and encryption.) In all of these cases, data elements are processed in a sequential order [15].

In the present paper, in order to reduce the computation overhead in signcryption algorithms, we use a pipelined technique which is chosen due to its suitability for Rasslan *et. al.* signcryption algorithm [5]. Rasslan *et. al.* signcryption algorithm is more efficient than all previously presented signcryption algorithms. It allows the recipient to recover the message blocks upon receiving their corresponding signature blocks, which makes Rasslan *et. al.* signcryption algorithm perfect for some application requirements. The proposed pipelined technique reduces the computation time required to execute Rasslan *et. al.* algorithm with respect to its corresponding values of a sequential execution. This is applied for any number of messages *'N'*.

This paper is organized as follows: in the next section, background and related work are detailed. In section 3, a description of Rasslan *et. al.* signcryption algorithm is given. Next, the proposed pipeline solution is discussed in section 4. Finally, the paper concludes in the last section.

2. Related Work

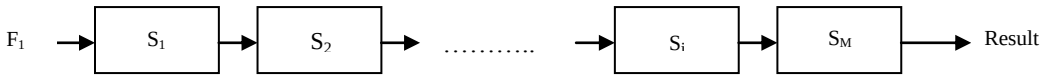
2.1 Pipelining cryptography

Many researchers have been working on accelerating the computation process of various cryptographic algorithms by using parallel and pipelined techniques. Some solutions were proposed to enhance the performance of symmetric encryption algorithms [8, 11-14]. Yang *et. al.* proposed a method to efficiently implement block cipher on Networked Processor Array (NePA) Network on Chip (NoP) platform using parallel and pipeline execution. They implement their solution for Data Encryption Standard (DES), Triple-DES algorithm, and Advanced Encryption Standard (AES). In addition, they proposed a new programming model resulting in a good performance with reduced development time. Agosta *et. al.* suggested a solution which is based on tile architectures, which allows high levels of instruction level parallelism. Alam *et. al.* proposed an architecture that is based on pipelined threads. Satoh achieved a high throughput for AES using pipelining of rounds. Abdellatif *et. al.* use pipelined technique to obtain high throughput for AES in Galois Counter Mode (GCM).

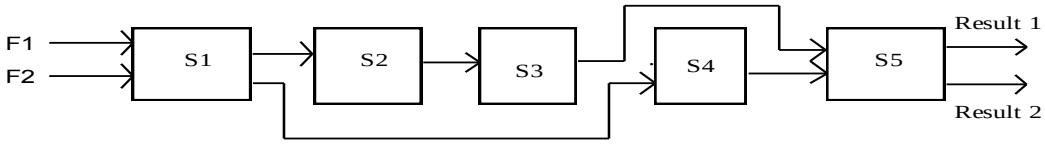
The public key cryptographic algorithms are computationally intensive, since it requires modular operations over large numbers. Several researches have been done to improve the performance of public encryption algorithms [6, 9-10, and 16]. Lin used Single Instruction Multiple Data (SIMD) technology to implement Rivest-Shamir-Adleman (RSA) public key encryption algorithm. To implement Elliptic Curve Cryptography (ECC) using pipelined technique, he achieved a high performance model with low power consumption. To implement Elliptic Curve Cryptography (ECC) using pipelined technique, Gutub proposed a scheme, which is efficient with respect to power, area, and throughput. Laue suggested a solution, which is based on the examination of the degree of parallelism on each abstraction level. He applied his solution to both RSA and ECC algorithms. GroBschadl suggested a solution, which is based on reducing the modular multiplication time, for the implementation of RSA algorithms. One of the most important issues in implementing public key encryption algorithms is to reduce the time of modular multiplication, since it is the most used in the majority of the public key encryption algorithms. In literature, many solutions were proposed [7, 10, and 17-19]. Mentens *et. al.*, in their work, presented a pipelined architecture of a modular Montgomery multiplier, which is suitable to be used in public key coprocessors. Their design makes use of 16-bit integer multiplication blocks that are available on recently manufactured Field Programmable Gate Arrays (FPGAs) [7]. Orton *et. al.* presented serial-parallel concurrent modular-multiplication architecture suitable for standard RSA encryption [17]. Gutub *et. al.* proposed pipelined cryptography modular multiplier architecture. That architecture was implemented on Field Programmable Gate Array (FPGA), designed in four stages to be properly suitable for elliptic curve crypto computation [18]. Meulenaer *et. al.* proposed an architecture that is based on a fully parallel and pipelined modular multiplier circuit. Meulenaer *et. al.* claimed that their architecture exhibited a 15-fold improvement over throughput/hardware cost ratio of previously published results [19]. GroBschadl *et. al.*, in their work [10], explained how three simple multiplications and one addition result in a modular multiplication, and how a modular exponentiation can be calculated by continued modular multiplications. In their work, they presented hardware algorithms for exponentiation, modular reduction and modular multiplication [10].

2.2 Parallel Pipelining

The pipeline pattern uses parallel tasks and concurrent queues to process a sequence of input values. Each task implements a stage of the pipeline. The queues act as buffers that allow the stages of the pipeline to be executed concurrently and in order. Pipelines can be considered as analogous to assembly lines in a factory, where each item in the assembly line is constructed in stages. The partially assembled item is passed from one assembly stage to another. The outputs of the assembly line occur in the same order as that of the input. Pipeline is the simplest and most fundamental architecture in parallelism. It can be a single function pipeline, or a multifunction pipeline. The single function pipeline precedes one operation on the stream of data, at each stage of the line. In the multifunction pipeline, different operations can be done at different stages. Fig. 1 shows both single and multifunction pipelines. In all pipelines, the data in the pipe is shifted from stage *"i"* to stage "*i*+1", at the same time, for all stages. At the shift time, all stages in the pipe must have completed their operation on their local data. The pipeline cycle time must not be less than the time required for the slowest stage to be completed. The execution unit consists of a number of stages, each of which performs a specific function within a specific time period. That is to say, a pipeline processor is the simplest and most fundamental architecture in parallel processing. In this model *"M"* processors are connected like an assembly line [20-25].



(a): A Single Functional Pipeline



(b): A Multifunction Pipeline

Fig. 1 The pipeline architecture.

3. Description of Rasslan *e.t al.* Signcryption Algorithm

In this section, a description of Rasslan *et. al.* signcryption algorithm is detailed [5]. This algorithm is more efficient than all the previously presented schemes. It allows the recipient (verifier) to recover the message blocks upon receiving their corresponding signature blocks. The scheme is perfect for some application requirements and it is designed for packet switched networks. In order to perform the proposed protocol, the following parameters must be set. First, the System Authority (SA) selects a large prime number p such that p-1 has a large prime factor q. SA also picks an integer, g, with order q in $GF(p)$. Let "f " be a secure one way hash function. SA publishes p, q, g and f. Each user, U_i, chooses a secret key $x_i \in Z_q$ and computes the corresponding public key $y_i = g^{x_i} \bmod p$. When a sender A wants to send a message to receiver B, it divides the stream into blocks of L packets ($Pack_1$, $Pack_2$, $Pack_3$,…, $Pack_{L-2}$, $Pack_{L-1}$, $Pack_L$). The value of these packets must be less than the value of p. The sender A, with secret key x_a and public key $y_a = g^{x_a}$, uses the following steps before sending the multicast message:

(1) Pick random numbers k, $l \in Z_q^*$ and set $r_0 = 0$, then compute $y_b^k \bmod p$ and $t = g^k \bmod p$.

(2) Compute: $r_i = Pack_i \,.\, f(r_{i-1} \oplus y_b^k) \bmod p$, for $i = 1, 2, \ldots, L$.

(3) Compute: $s = k - r \,.\, x_a \bmod q$, where $r = f(r_1, r_2, r_3, \ldots, r_L)$.

(4) Then, the sender computes $c_1 = g^l \bmod p$ and $c_2 = r_L \,.\, y_b^l \bmod p$.

After receiving the sent message, the recipient checks the signature by comparing t^{x_a} to ($y_b^s \,.\, y_{ab}^r \bmod p$), where $y_{ab} = y_a^{x_b} \bmod p$. If the check doesn't hold, this indicates that the received packets are modified and must be discarded. On the other hand, if the check holds, then each recipient calculates $r_L = c_2 \,.\, c_1^{-x_b} \bmod p$. Finally, each recipient recovers message blocks using the following equation: $Pack_i = r_i \,.\, f(r_{i-1} \oplus t^{x_b})^{-1} \bmod p$, for $i = 1, 2, \ldots, L$ and $r_0 = 0$. One advantage of the proposed protocol, since it is based on signcryption techniques, is that it provides both confidentiality and authenticity in one step. Consequently, the computation overhead decreases, this makes the proposed protocol suitable for real-time applications. In the next section, the proposed pipelined technique is detailed.

4. Proposed Pipeline Design for Rasslan *et. al.* Signcryption Algorithm

Pipelining is based on breaking a task into steps performed by different processor units with inputs streaming through. It is much like an assembly line. Due to the nature of Rasslan *et. al.* algorithm, which is characterized by repeating the same function for several packets, we decided to use single-function multiple-input architecture. The proposed technique uses *'M''* processors/stages to perform the signcryption operation. Assuming that the data stream is divided into '*N*' messages, where each message contains '*L*' packets and the number of packets equals to the number of functions/tasks to be executed. Each packet *'$Pack_{ij}$'*, (where *i* ranges from 1 to *L* and *j* ranges from 1 to *N*), is passed to the corresponding function unit ($Pack_{ij} \,.\, f(r_{i-1j} \oplus y_b^k) \bmod p$). The output of each function unit is shifted from stage *i* to stage *i*+1 for all stages, at the same time, as shown in Fig. 2. In our proposed design, there are three cases:

(i) Single processor – Single task,
(ii) Multiple processors – Single task
(iii) Single processor – Multiple tasks

4.1 Single processor – Single task

In this case, the number of processors equals to the number of function units per tasks to be executed (M=L). Assume

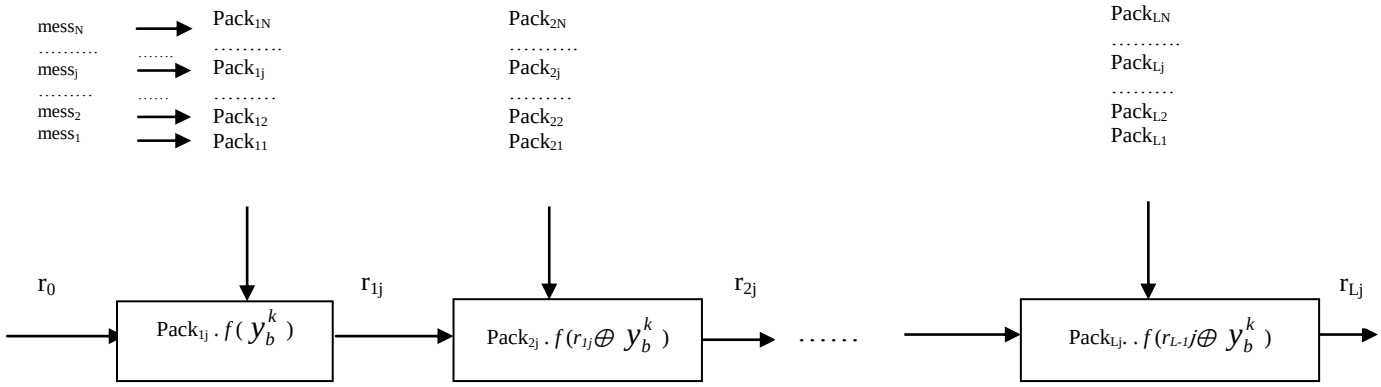


Fig. 2 Steps of Rasslan *et. al.* algorithm.

that each function unit needs '*T*' units of time to be processed. Therefore, the sequential time *'T_s'* to execute '*N*' messages, each of '*L*' packets, is given by the following equation:

$$T_S = N*L*T \quad (1)$$

In case of *M*=*L*, the total time to compute '*N*' messages in parallel (using pipelined architecture) is given by the following equation:

$$T_{par} = (N+L-1)*T \quad (2)$$

Compared to the non-pipelined model, the speed-up *'S_p'* is given by:

$$S_p = \left\lfloor \frac{N*L}{(N+L-1)} \right\rfloor \quad (3)$$

In addition, the overall efficiency *'E_p'* equals to:

$$E_p = \left\lfloor \frac{N*L}{(N+L-1)} \right\rfloor * \frac{1}{M} \quad (4)$$

Moreover, it improves the total computation time of Rasslan *et. al.* algorithm by:

$$\left\lfloor \frac{T_s - T_{par}}{T_s} \right\rfloor = \left\lfloor \frac{(N-1)*(L-1)}{(N*L)} \right\rfloor \quad (5)$$

Fig. 3 shows T_{par} for different cases of *N* and *L*. The first case is N = L. The second case is L < N. The last case is L > N. This figure shows that T_{par} is not affected by the relation between N and L, for all cases.

4.2 Multiple processors – Single task

In this case, M > L. When assigning one task per function unit, to each processor, only *'L'* processors are needed and *'M-L'* processors are idle. This leads to load imbalance. In this case only '*L*' processors will be used for pipelining and T_{par}= ((*N*+*L*-*1*)***T*), as discussed above. To avoid load imbalance, more than one processor can cooperate to compute different instructions of each function unit ($Pack_{ij}$. $f(r_{i-1j} \oplus y_b^k)$). This task can be divided into three subtasks. The first task is to execute the modular exponentiation. This task is calculated once for each message. Therefore, it will be performed sequentially and will not be considered in our calculations. The second task is the execution of the XOR operation. The time to carry out this operation is relatively very small compared to the modular multiplication and exponentiation. Hence, it will be neglected in our calculations. Finally, the third task is to compute the modular multiplication operation. Thus, we consider the time needed to compute a single function unit '*T*' is equal to the time needed to calculate one modular multiplication function. As shown in [10], a modular multiplication can be divided into three simple multiplications and one addition. The modular multiplication task is the most consuming time. Therefore, our objective is to reduce the modular multiplication time. This can be achieved by incorporating the idle processors in the execution of each modular multiplication operation. Each modular multiplication task is represented as shown in Fig. 4. It can be computed by three processors in parallel, as shown in Fig. 5. Let the time needed to compute a simple multiplication operation equals to "t_p" and the time needed for computing simple addition operation equals to "t_a". Since, the addition operation considerably needs less time than the multiplication

F_1	$mess_1$ $Pack_1$	$mess_2$ $Pack_1$	$mess_3$ $Pack_1$	$mess_4$ $Pack_1$	$mess_5$ $Pack_1$	$mess_6$ $Pack_1$					
F_2		$mess_1$ $Pack_2$	$mess_2$ $Pack_2$	$mess_3$ $Pack_2$	$mess_4$ $Pack_2$	$mess_5$ $Pack_2$	$mess_6$ $Pack_2$				
F_3			$mess_1$ $Pack_3$	$mess_2$ $Pack_3$	$mess_3$ $Pack_3$	$mess_4$ $Pack_3$	$mess_5$ $Pack_3$	$mess_6$ $Pack_3$			
F_4				$mess_1$ $Pack_4$	$mess_2$ $Pack_4$	$mess_3$ $Pack_4$	$mess_4$ $Pack_4$	$mess_5$ $Pack_4$	$mess_6$ $Pack_4$		
F_5					$mess_1$ $Pack_5$	$mess_2$ $Pack_5$	$mess_3$ $Pack_5$	$mess_4$ $Pack_5$	$mess_5$ $Pack_5$	$mess_6$ $Pack_5$	
F_6						$mess_1$ $Pack_6$	$mess_2$ $Pack_6$	$mess_3$ $Pack_6$	$mess_4$ $Pack_6$	$mess_5$ $Pack_6$	$mess_6$ $Pack_6$
Time	T	2T	3T	4T	5T	6T	7T	8T	9T	10T	11T

(a) L = N and N= 6

F_1	$mess_1$ $Pack_1$	$mess_2$ $Pack_1$	$mess_3$ $Pack_1$	$mess_4$ $Pack_1$	$mess_5$ $Pack_1$	$mess_6$ $Pack_1$	$mess_7$ $Pack_1$					
F_2		$mess_1$ $Pack_2$	$mess_2$ $Pack_2$	$mess_3$ $Pack_2$	$mess_4$ $Pack_2$	$mess_5$ $Pack_2$	$mess_6$ $Pack_2$	$mess_7$ $Pack_2$				
F_3			$mess_1$ $Pack_3$	$mess_2$ $Pack_3$	$mess_3$ $Pack_3$	$mess_4$ $Pack_3$	$mess_5$ $Pack_3$	$mess_6$ $Pack_3$	$mess_7$ $Pack_3$			
F_4				$mess_1$ $Pack_4$	$mess_2$ $Pack_4$	$mess_3$ $Pack_4$	$mess_4$ $Pack_4$	$mess_5$ $Pack_4$	$mess_6$ $Pack_4$	$mess_7$ $Pack_4$		
F_5					$mess_1$ $Pack_5$	$mess_2$ $Pack_5$	$mess_3$ $Pack_5$	$mess_4$ $Pack_5$	$mess_5$ $Pack_5$	$mess_6$ $Pack_5$	$mess_7$ $Pack_5$	
F_6						$mess_1$ $Pack_5$	$mess_2$ $Pack_6$	$mess_3$ $Pack_6$	$mess_4$ $Pack_6$	$mess_5$ $Pack_6$	$mess_6$ $Pack_6$	$mess_7$ $Pack_6$
Time	T	2T	3T	4T	5T	6T	7T	8T	9T	10T	11T	12T

(b) N >L and N=7

F_1	$mess_1$ $Pack_1$	$mess_2$ $Pack_1$	$mess_3$ $Pack_1$	$mess_4$ $Pack_1$	$mess_5$ $Pack_1$					
F_2		$mess_1$ $Pack_2$	$mess_2$ $Pack_2$	$mess_3$ $Pack_2$	$mess_4$ $Pack_2$	$mess_5$ $Pack_2$				
F_3			$mess_1$ $Pack_3$	$mess_2$ $Pack_3$	$mess_3$ $Pack_3$	$mess_4$ $Pack_3$	$mess_5$ $Pack_3$			
F_4				$mess_1$ $Pack_4$	$mess_2$ $Pack_4$	$mess_3$ $Pack_4$	$mess_4$ $Pack_4$	$mess_5$ $Pack_4$		
F_5					$mess_1$ $Pack_5$	$mess_2$ $Pack_5$	$mess_3$ $Pack_5$	$mess_4$ $Pack_5$	$mess_5$ $Pack_5$	
F_6						$mess_1$ $Pack_5$	$mess_2$ $Pack_6$	$mess_3$ $Pack_6$	$mess_4$ $Pack_6$	$mess_5$ $Pack_6$
Time	T	2T	3T	4T	5T	6T	7T	8T	9T	10T

(c) N<L and N=5

Fig. 3 The total execution time after using pipelining for M = L =6

operation, the time "t_a" can be neglected. Then, $T= 3t_p$. Given that the modular multiplication can be done through using three simple multiplications, the maximum improvement in the execution time of the modular multiplication could be achieved through using three single processors. That is to say, in our case, the optimal number of processors to execute one function unit is three. For each function unit, we need two processors to help each overloaded processor. This means that to avoid load imbalance, two volunteer processors are needed to help

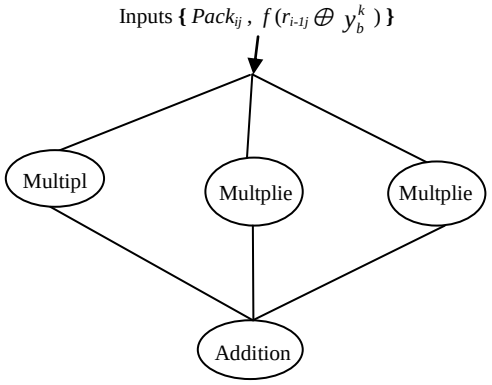


Fig. 4 Representation of modular multiplication.

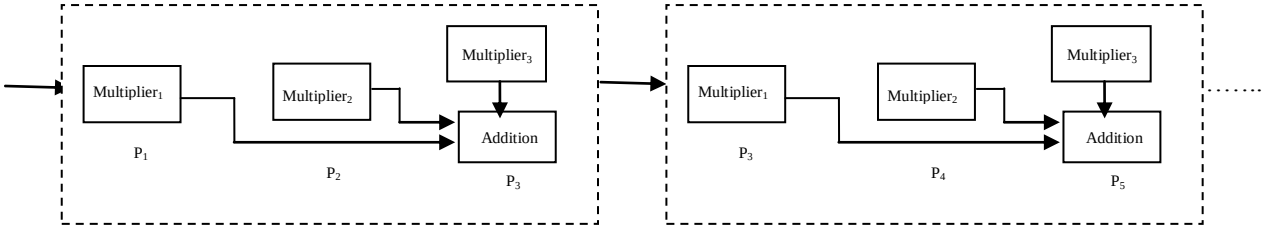


Fig. 5 Distribution of tasks in case of multiple processors – single task.

each overloaded processor to finish its work. In case of the number of volunteer processors is less than two, the waiting time will increase. To start load balancing, (M-L) must be greater than two. On the other hand, if the number of volunteers is greater than two, more than one processor will cooperate on the execution of each simple multiplication (fine grained parallelization). Due to the dependency in the execution of simple multiplication, the communication overhead increases. Thus, we will only consider the case of two volunteer processors to help in each function unit. Hence, we have two cases:
$\left(\frac{M-L}{2}\right) \geq 3$ and $\left(\frac{M-L}{2}\right) < 3$.

Case 1:

$$\left|\frac{M-L}{2}\right| \geq 3$$

In this case, the total time to compute 'N' messages in parallel (using pipelined architecture) is given by the following equation:

$$\text{Tpar} = \left\{\left[(N+L) - \left|\frac{M-L}{2}\right|\right] * T + \left[\left|\frac{M-L}{2}\right| - 3\right] * t_p\right\} \quad (6)$$

The speed-up 'S_p' is given by:

$$\text{Sp} = \frac{(N*L*T)}{\left\{\left[(N+L) - \left|\frac{M-L}{2}\right|\right] * T + \left[\left|\frac{M-L}{2}\right| - 3\right] * t_p\right\}} \quad (7)$$

In addition, the overall efficiency 'E_p' equals to:

$$\text{Ep} = \frac{(N*L*T)}{\left\{\left[(N+L) - \left|\frac{M-L}{2}\right|\right] * T + \left[\left|\frac{M-L}{2}\right| - 3\right] * t_p\right\}} * \frac{1}{M} \quad (8)$$

Moreover, it improves the total computation time of Rasslan *et. al.* algorithm by:

Improvement w.r.t sequential=

$$\frac{(N*L*T) - \left\{\left[(N+L) - \left|\frac{M-L}{2}\right|\right] * T + \left[\left|\frac{M-L}{2}\right| - 3\right] * t_p\right\}}{(N*L*T)} \quad (9)$$

Improvement w.r.t before balance=

$$\frac{((N+L-1)*T) - \left\{\left[(N+L) - \left|\frac{M-L}{2}\right|\right] * T + \left[\left|\frac{M-L}{2}\right| - 3\right] * t_p\right\}}{(N+L-1)*T} \quad (10)$$

Fig. 6 presents *Tpar* for M > L, and $\left|\frac{M-L}{2}\right| \geq 3$. As shown in the previous subsection, the relation between N and L will not affect the total execution time *Tpar*, thus we use only N=L=6 as an example.

Case 2: $\left|\frac{M-L}{2}\right| < 3$

In this case, the total time to compute 'N' messages in parallel (using pipelined architecture) is given by the following equation:

$$\text{Tpar} = \left\{\left[(N+L-1) - \left|\frac{M-L}{2}\right|\right] * T + \left|\frac{M-L}{2}\right| * t_p\right\} \quad (11)$$

The speed-up 'S_p' is given by:

$$\text{Sp} = \frac{(N*L*T)}{\left\{\left[(N+L-1) - \left|\frac{M-L}{2}\right|\right] * T + \left|\frac{M-L}{2}\right| * t_p\right\}} \quad (12)$$

In addition, the overall efficiency 'E_p' equals to:

$$\text{Ep} = \frac{(N*L*T)}{\left\{\left[(N+L-1) - \left|\frac{M-L}{2}\right|\right] * T + \left|\frac{M-L}{2}\right| * t_p\right\}} * \frac{1}{M} \quad (13)$$

Moreover, it improves the total computation time of Rasslan *et. al.* algorithm by:

Improvement w.r.t sequential =

$$\frac{(N*L*T) - \left\{\left[(N+L-1) - \left|\frac{M-L}{2}\right|\right] * T + \left|\frac{M-L}{2}\right| * t_p\right\}}{(N*L*T)} \quad (14)$$

Improvement w.r.t before balance =

$$\frac{((N+L-1)*T) - \left\{\left[(N+L-1) - \left|\frac{M-L}{2}\right|\right] * T + \left|\frac{M-L}{2}\right| * t_p\right\}}{(N+L-1)*T} \quad (15)$$

Fig. 7 illustrates T_{par} for M > L, and $\left|\frac{M-L}{2}\right| < 3$.

4.3 Single processor – Multiple tasks

In this case, where M < L, to achieve the optimum processor utility, each processor can execute one or more consecutive tasks. Then, the first processor gives its output to the subsequent processor. This is repeated for the different processors as shown in Fig. 8. In the following paragraphs, the total parallel (pipelined) time T_{par} is calculated for the following two cases:

- First, for (L mod M) = 0, where each processor computes $\left(\frac{L}{M}\right)$ tasks.
- Second, for (L mod M) ≠ 0, where each processor of the first (L mod M) processors computes $\left|\frac{L}{M}\right|$ +1 tasks, while each processor of the remaining processors executes $\left|\frac{L}{M}\right|$ tasks.

Case 1: (L mod M) = 0

In this case, the total time to compute 'N' messages in parallel (using pipelined architecture) is given by the following equation:

$$\text{Tpar} = \left((N-1) * \left(\frac{L}{M}\right) + L\right) * T \quad (16)$$

The speedup 'S_p' is given by:

$$S_p = \frac{N*L}{\left((N-1) * \left(\frac{L}{M}\right) + L\right)} \quad (17)$$

In addition, the overall efficiency 'E_p' equals to:

$$\text{Ep} = \frac{N*L}{\left((N-1) * \left(\frac{L}{M}\right) + L\right)} * \frac{1}{M} \quad (18)$$

Improvement w.r.t sequential=

$$\frac{(N*L) - \left((N-1) * \left(\frac{L}{M}\right) + L\right)}{(N*L)} \quad (19)$$

Improvement w.r.t before balance=

$$\frac{(N+L-1) - \left((N-1) * \left(\frac{L}{M}\right) + L\right)}{(N+L-1)} \quad (20)$$

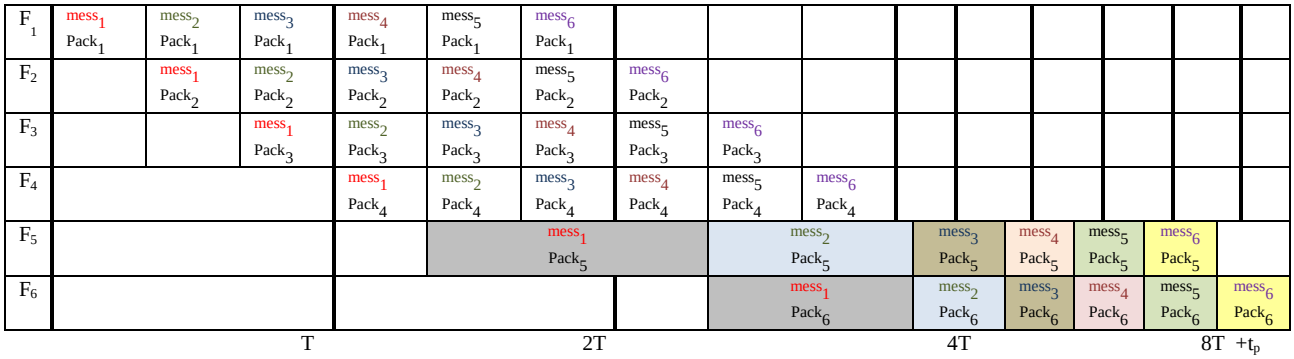


Fig. 6 The total execution time after using pipelining for $\left|\frac{M-L}{2}\right| \geq 3$, N=L=6 and M =15.

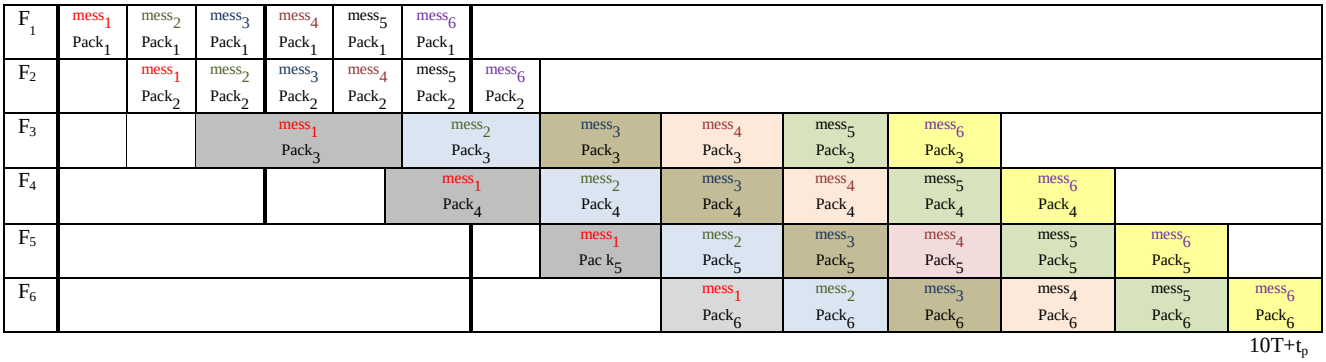


Fig. 7 The total execution time after using pipelining for $\left|\frac{M-L}{2}\right| < 3$, N=L=6 and M=9.

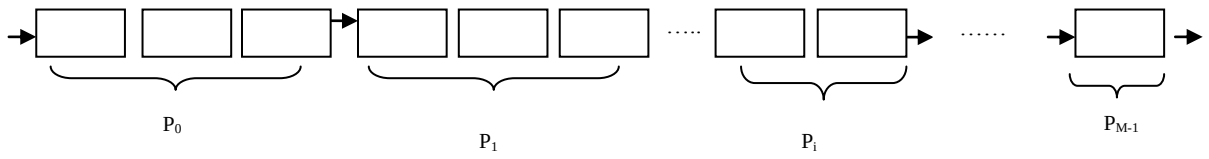


Fig. 8 Distribution of tasks in case of single processor – multiple tasks.

Case 2: (L mod M) $\neq$ 0

In this case, the total time to compute 'N' messages in parallel (using pipelined architecture) is given by the following equation:

$$\text{Tpar} = \left\{\left[(N-1)*\left(\left|\frac{L}{M}\right|+1\right)+(L \bmod M)+\left(M*\left|\frac{L}{M}\right|\right)\right]*T\right\} \quad (21)$$

$$\text{Tpar} = \left((N-1)*\left(\left|\frac{L}{M}\right|+1\right)+L\right)*T$$

The speed-up 'S_p' is given by:

$$\text{Sp} = \frac{N*L}{\left((N-1)*\left(\left|\frac{L}{M}\right|+1\right)+L\right)} \quad (22)$$

In addition, the overall efficiency 'E_p' equals to:

$$\text{Ep} = \frac{N*L}{\left((N-1)*\left(\left|\frac{L}{M}\right|+1\right)+L\right)} * \frac{1}{M} \quad (23)$$

P_0	$mess_1$ $Pack_1$	$mess_1$ $Pack_2$	$mess_2$ $Pack_1$	$mess_2$ $Pack_2$	$mess_3$ $Pack_1$	$mess_3$ $Pack_2$	$mess_4$ $Pack_1$	$mess_4$ $Pack_2$	$mess_5$ $Pack_1$	$mess_5$ $Pack_2$	$mess_6$ $Pack_1$	$mess_6$ $Pack_2$				
P_1			$mess_1$ $Pack_3$	$mess_1$ $Pack_4$	$mess_2$ $Pack_3$	$mess_2$ $Pack_4$	$mess_3$ $Pack_3$	$mess_3$ $Pack_4$	$mess_4$ $Pack_3$	$mess_4$ $Pack_4$	$mess_5$ $Pack_3$	$mess_5$ $Pack_4$	$mess_6$ $Pack_3$	$mess_6$ $Pack_4$		
P_2					$mess_1$ $Pack_5$	$mess_1$ $Pack_6$	$mess_2$ $Pack_5$	$mess_2$ $Pack_6$	$mess_3$ $Pack_5$	$mess_3$ $Pack_6$	$mess_4$ $Pack_5$	$mess_4$ $Pack_6$	$mess_5$ $Pack_5$	$mess_5$ $Pack_6$	$mess_6$ $Pack_5$	$mess_6$ $Pack_6$
	T			4T				8T				12T		16T		

Fig. 9 The total execution time after using pipelining for (L mod M) =0, M=3 and N= L =6.

P_0	$mess_1$ $Pack_1$	$mess_1$ $Pack_2$	$mess_2$ $Pack_1$	$mess_2$ $Pack_2$	$mess_3$ $Pack_1$	$mess_3$ $Pack_2$	$mess_4$ $Pack_1$	$mess_4$ $Pack_2$	$mess_5$ $Pack_1$	$mess_5$ $Pack_2$	$mess_6$ $Pack_1$	$mess_6$ $Pack_2$				
P_1			$mess_1$ $Pack_3$		$mess_2$ $Pack_3$		$mess_3$ $Pack_3$		$mess_4$ $Pack_3$		$mess_5$ $Pack_3$		$mess_6$ $Pack_3$			
P_2				$mess_1$ $Pack_4$		$mess_2$ $Pack_4$		$mess_3$ $Pack_4$		$mess_4$ $Pack_3$		$mess_5$ $Pack_4$		$mess_6$ $Pack_4$		
P_3					$mess_1$ $Pack_5$		$mess_2$ $Pack_5$		$mess_3$ $Pack_5$		$mess_4$ $Pack_5$		$mess_5$ $Pack_5$		$mess_6$ $Pack_5$	
P_4						$mess_1$ $Pack_6$		$mess_2$ $Pack_6$		$mess_3$ $Pack_6$		$mess_4$ $Pack_6$		$mess_5$ $Pack_6$		$mess_6$ $Pack_6$
	T			4T				8T				12T				16T

Fig. 10 The total execution time after using pipelining for (L mod M)≠0, L=N = 6 and M=5.

Improvement w.r.t sequential =

$$\frac{(N*L)-\left((N-1)*\left(\left|\frac{L}{M}\right|+1\right)+L\right)}{(N*L)} \quad (24)$$

Improvement w.r.t before balance =

$$\frac{(N+L-1)-\left((N-1)*\left(\left|\frac{L}{M}\right|+1\right)+L\right)}{(N+L-1)} \quad (25)$$

Fig. 9 shows T_{par} for (L mod M) =0, while Fig. 10 shows T_{par} for (L mod M) ≠0.

4.4 Discussion of results

Signcryption algorithmss suffer from the high computation overhead. In the present paper, we propose a pipeline design to solve this problem. The proposed technique is analyzed according to the following aspects: the parallel time '*Tpar*', the speed-up '*Sp*', the efficiency '*Ep*', and the degree of improvement. Figures 11-13 show the system performance: ' *Tpar*', ' *Sp*', ' *Ep*', and the improvement degree. Fig. 11 shows the system performance in case of single processor – single task. While, Fig. 12 illustrates the performance in case of multiple processors – single task. Finally, Fig. 13 presents the system performance in case of single processor – multiple tasks. From the above figures, the following observations are noted:

- The proposed pipelined technique reduces the computation time required to execute Rasslan *et. al.* algorithm, compared to its corresponding values of sequential execution. This is applied for any number of messages *'N'*.
- For the case of single processor – single task (M=L), as the number of messages increases, the usefulness of using parallel computing increases. This is evident in enhancing the system performance as illustrated in Fig. 11. The degree of improvement of the proposed technique, compared to the performance prior to parallelization is 62.5%, 71.4%, 74.1%, 76.9% and 77.7%, for L =M = 6 and N = 4, 7,9,13 and 15, respectively.
- For the case of multiple processors – single task (M>L), when the number of processors increases, the total execution time decreases and consequently the efficiency will decrease, as illustrated in Fig. 12. The degree of improvement of the proposed technique, compared to the performance of Rasslan *et. al.* algorithm without using pipelining (serial), is 71.3%, 73.2%, 75%, and 76.8% for L =N = 6 and M = 8,10,13, and 15 respectively. On the other hand, the degree of improvement, compared to the performance prior to balancing is 6%, 12.2%, 18.2%, and 24.3% for the M= 8,10,13, and 15 respectively. The performance of the proposed pipelined design is better than both cases (without pipelining and before balance.)
- In the case of single processor – multiple tasks (M<L), as the number of processors increases, the system performance enhances. This is shown in Fig. 13. This figure illustrates that for some values where $\left|\frac{L}{M}\right|$ is constant, the total execution time is constant until it saturates at $\left|\frac{L}{M}\right|$=1. Therefore, in this case, the

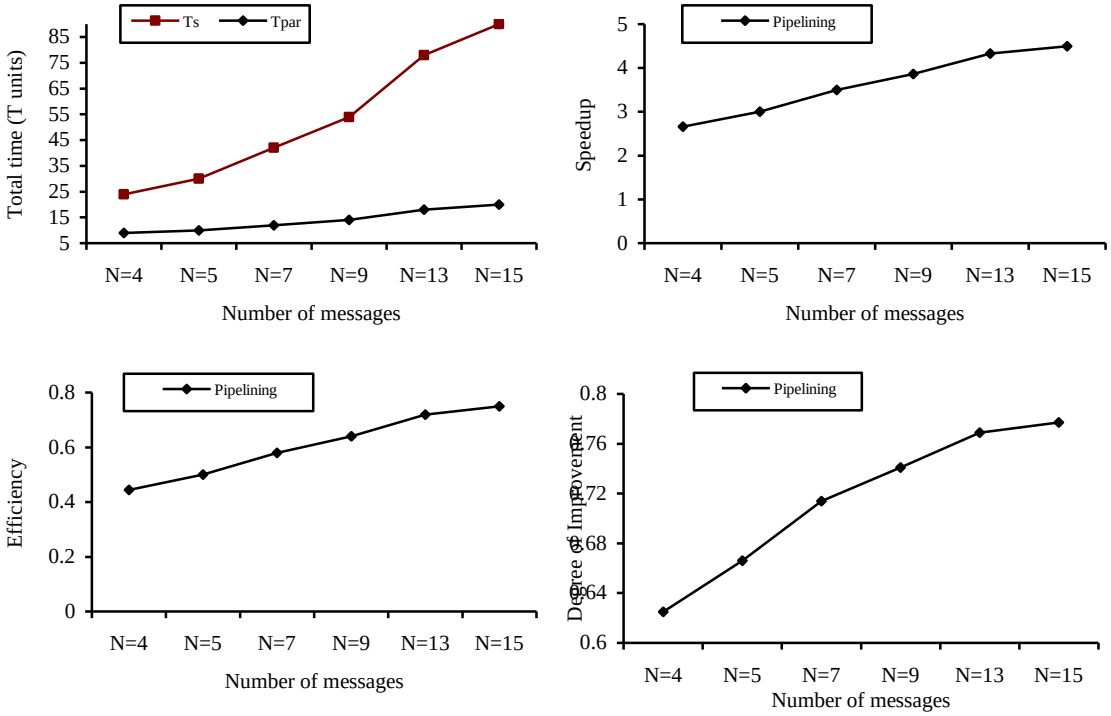


Fig. 11 M=L=6, for different values of N.

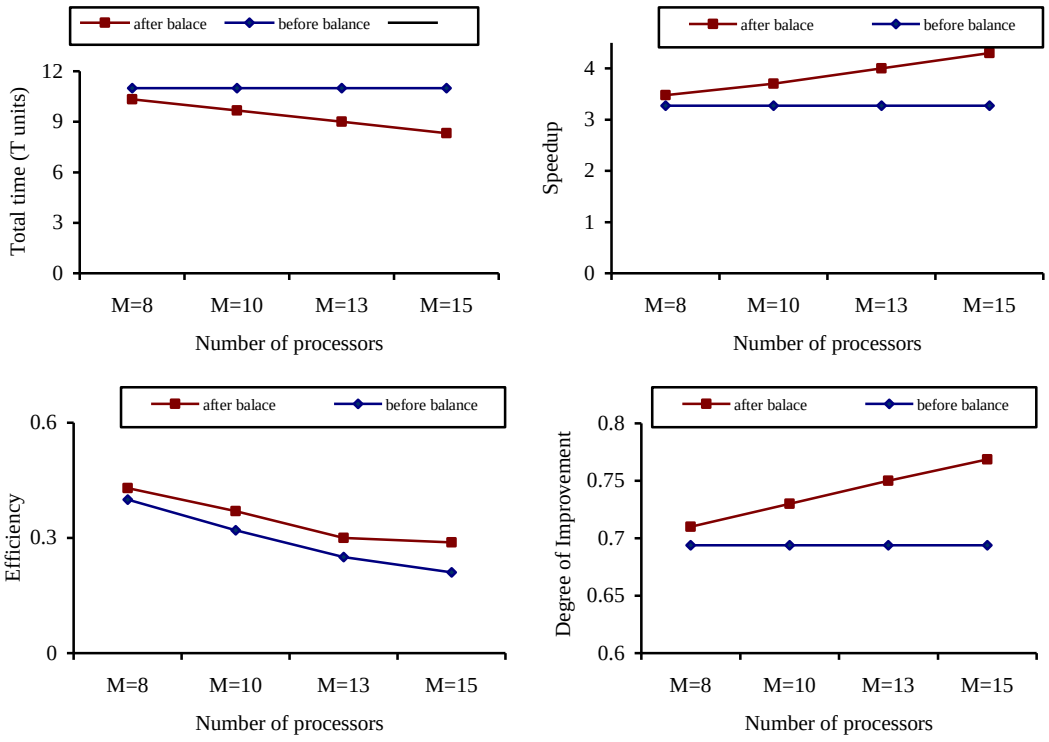


Fig. 12 M>L, L=N=6, for different values of M.

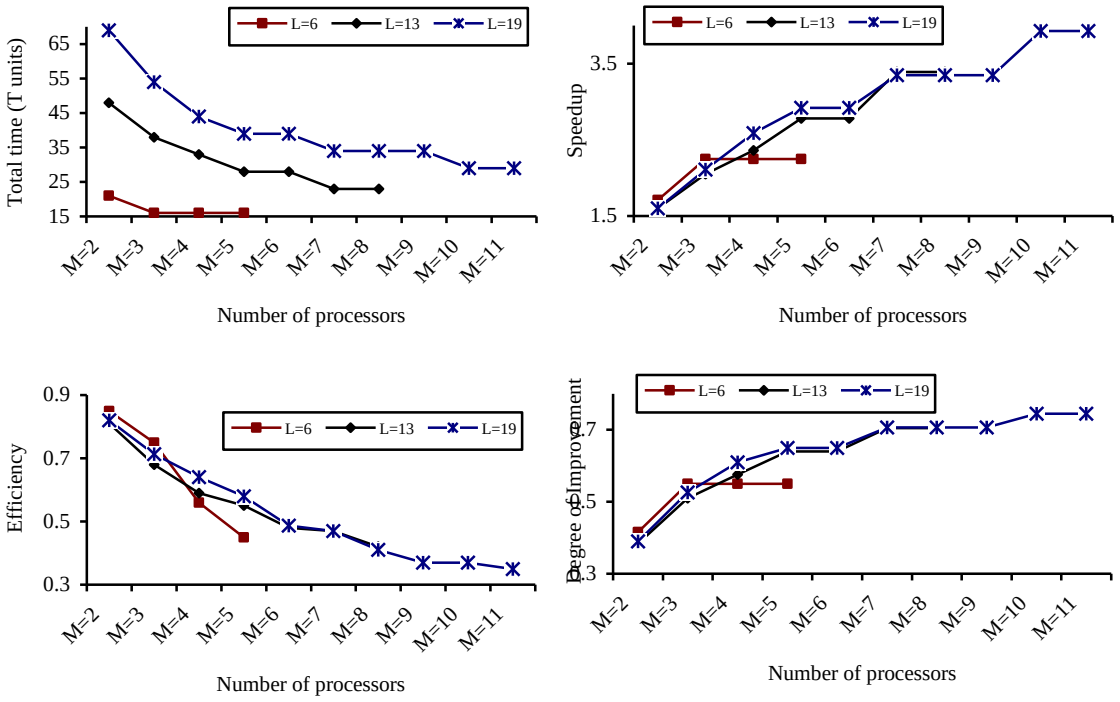


Fig. 13 M<L, N=6, for different values of L and M.

maximum number of processors should not exceed $\frac{'L'}{2}$. Fig. 13 shows that for L=19, N=6, the degree of improvement of the proposed technique is 39%, 61%, 65%, 70% and 74%, and for M = 2,4,6,8, and 10, respectively. For the same number of messages (and L=13), the degree of improvement is 38.4%, 57.61%, 64%, and 70%, and for M = 2, 4, 6, and 8, respectively. Otherwise, for L =N= 6, the degree of improvement is 41.6%, 55%, and 55%, and for M = 2, 4, and 6, respectively.

5. Conclusions

In the present paper, we address the problem of computation in signcryption techniques. We have chosen Rasslan *et. al.* signcryption algorithm over other signcryption algorithms, because it has lower computation and communication overheads than all previously presented algorithms. However, Rasslan *et. al.* algorithm still suffers from relatively high computation overhead. To enhance the performance of Rasslan *et. a.l* algorithm, we use a pipelined technique to speed-up the arithmetic operations. Therefore, the total computation time is reduced. Furthermore, we evaluate the performance of the proposed solution with other techniques. The results show that the proposed pipelined technique reduces the computation time required to execute Rasslan *et. al.* algorithm, compared to its corresponding values of sequential execution. This is applied for any number of messages *'N'*. In our proposed design, there are three cases: (i) Single processor – Single task (M=L), (ii) Multiple processors – Single task (M>L), and (iii) Single processor – Multiple tasks (M<L). For the first case, we assume that L =M = 6. The degree of improvement of the proposed technique, compared to the performance prior to parallelization, is 62.5%, 71.4%, 74.1%, 76.9% and 77.7%, assuming that N = 4, 7,9,13, and 15 respectively. For the case of multiple processors – single task, the degree of improvement, compared to the performance of Rasslan *et. al.* algorithm without using pipelining is 71.3%, 73.2%, and 75% assuming that L = N =6, and M= 8,10,13, and 15, respectively. Finally, for the last case, we assume that L =19, N=6, the degree of improvement of the proposed technique is 39%, 61%, 65%, 70% and 74%, for M = 2,4,6,8, and 10, respectively.

References

[1] Y. Zheng , "Digital Signcryption or How to Achieve Cost (Signature & Encryption) Cost (Signature) + Cost (Encryption)", International Conference of CRYPTO'97, LNCS 1294, Springer-Verlag, pp. 165-179, 1997.

[2] C. K. Li and D. S. Wong, "Signcryption from Randomness Recoverable Public Key Encryption", Inform. Sci., vol. 180, pp.549-559, 2010.

[3] L. Pang, H. Li, L. Gao and Y. Wang , "Completely Anonymous Multi-Recipient Signcryption Scheme with

Public Verification", PLos ONE, vol. 8, No. 5, pp. 1-10, 2013.

[4] F. Li, X. Xin and Y. Hu, "Identity-Based Broadcast Signcryption", Computer Standards and Interfaces, vol. 30, pp. 89-94, 2008.

[5] M. Rasslan and H. Aslan, "On the Security of Two Improved Authenticated Encryption Schemes", Accepted to be published at Int. J. of Security and Networks, vol. 8, no. 4, pp. 194-199, 2013.

[6] B. Lin, "Solving Sequential Problems in Parallel: An SIMD Solusion of RAS Cryptography", freescale Semiconductor, Inc., 2004, 2006.

[7] N. Mentens, K. Sakiyama, B. Preneel and I. Verbauwhede," Efficient Pipelining for Modular Multiplication Architectures in Prime Fields", International Conference of GLSVLSI'07, Stresa-Lago Maggiore, Italy, pp. 11-13, March 2007.

[8] Y. Yang, J. Bahn, S. Lee and N. Bagherzadeh," Parallel and Pipeline Processing for Block Cipher Algorithms on a Network –on- Chip", Sixth International Conference on Information Technology: New Generations, IEEE Computer Society, pp. 849-854, 2009.

[9] A. A. Gutub, "Merging GF(p) Elliptic Curve Point Adding and Doubling on Pipelined VLSI Cryptographic ASIC Architecture", International Journal of Computer Science and Network Security (IJCSNS), vol.6, no.3A, pp. 44-52, March, 2006.

[10] J. GroBschadl, "High-Speed RSA Hardware Based on Barret's Modular Reduction Method", C_ .K. Ko_c and C. Paar (Eds.): CHES 2000, LNCS 1965, 2000, pp. 191-203, © Springer-Verlag Berlin Heidelberg 2000.

[11] G. Agosta, L. Breveglieri, G. Pelosi, M. Sykora, and P. di Milano," Programming Highly Parallel Reconfigurable Architectures for Symmetric and Asymmetric Cryptographic Applications ", Journal of Computers , vol. 2, no. 9, pp. 50-59, Nov., 2007.

[12] M. Alam and W. B., and G. Jullien, "A Novel Pipelined Threads Architecture for AES Encryption Algorithm", International Conference on Application-Specific Systems, Architectures, and Processors (ASAP'02), IEEE Computer Society, Washington, DC, USA, page 296, 2002.

[13] K. M. Abdellatif, R. Chotin-Avot, and H. Mehrez, "Efficient Parallel-Pipelined GHASH for Message Authentication", International Conference on ReConFigurable Computing and FPGAs, Cancun, Quintana Roo, Mexico, pp. 1-6, IEEE, 2012.

[14] A. Satoh, "High- Speed Hardware Architectures for Authenticated Encryption Mode GCM", IEEE International Symposium on Circuits and Systems (ISCAS), Island of Kos, Greece, pp 4831-4834, 21-24 May, 2006.

[15] http://msdn.microsoft.com/enus/library/ff963548.aspx.

[16] R. Laue, "Efficient and Flexible Co-processor for Server-Based Public Key Cryptography Applications", Secure Embedded Systems, LNEE 78, A. Biedermann and H. Gregor Molter (Eds.): springerlink.com © Springer-Verlag Berlin Heidelberg, pp. 129–149, 2010.

[17] G. Orton, Lloyd Peppard, and Stafford Tavares," A Design of a Fast Pipelined Modular Multiplier Based on a Diminished-Radix Algorithm", Journal of Cryptology, vol. 6, pp.183-208, 1993.

[18] A. Gutub, A. El-Shafei and M. Aabed", Implementation of a pipelined modular multiplier architecture for GF(p) elliptic curve cryptography computation", Kuwait Journal of Science and Engineering, vol. 38(2B), pp. 125-153, 2011.

[19] G. de Meulenaer, F. Gosset, G. Meurice de Dormale, and J.-J. Quisquater, "Integer Factorization Based on Elliptic Curve Method: Towards Better Exploitation of Reconfigurable Hardware", 15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM), pp. 197-206, Napa-CA, 23-25 April, 2007.

[20] S. G. Aki, Parallel Computation: *Models and Methods*, Prentice-Hall, Inc., 1997.

[21] Chalmers, and J. Tidmus, *Practical Parallel Processing: An Introduction to Problem Solving in Parallel*, International Thomson Computer Press,1996.

[22] M. J. Flynn, *Computer Architecture: Pipelined and Parallel Processor Design*, Jones & Bartlett Learning Publications, 1995.

[23] D. Moldovan, *Parallel Processing from Applications to Systems*, Morgan Kaufmann Publishers, 1993.

[24] E. V. Krishnamurthy, *Parallel Processing Principles and Practice*, Addison-Wesley Publishing Company, Inc., 1989.

[25] C.V. Ramanoorthy, and H. F. Li, "Pipeline Architecture", Computer Surveys, vol. 9, no. 1, pp. 61-103, March 1977.

Ghada F. ElKabbany is an Assistant Professor at Electronics Research Institute, Cairo- Egypt. She received her B.Sc. degree, M.Sc. degree and Ph.D. degree in Electronics and Communications Engineering from the Faculty of Engineering, Cairo University, Egypt. Her research interests include High Performance Computing (HPC), Robotics, and Computer Network Security.

Heba K. Aslan is an Associate Professor at Electronics Research Institute, Cairo-Egypt. She received her B.Sc. degree, M.Sc. degree and Ph.D. degree in Electronics and Communications Engineering from the Faculty of Engineering, Cairo University, Egypt in 1990, 1994 and 1998 respectively. Aslan has supervised several masters and Ph.D. students in the field of computer networks security. Her research interests include: Key Distribution Protocols, Authentication Protocols, Logical Analysis of Protocols and Intrusion Detection Systems.

Mohamed N. Rasslan: is an Assistant Professor at Electronics Research Institute, Cairo, Egypt. He received the B.Sc., M.Sc., degrees from Cairo University and Ain Shams University, Cairo, Egypt, in 1999 and 2006 respectively, and his his Ph.D. from Concordia University, Canada 2010. His research interests include: Cryptology, Digital Forensics, and Networks Security.