New Generic Design to Expedite Asymmetric Cryptosystems Using Three-levels of Parallelism

Mohamed Rasslan[1], Ghada Elkabbany[2], and Heba Aslan[3]
(Corresponding author: Mohamed Raslan)
Informatics Dept., Electronics Research Institute
National Research Center building, El Tahrir, St. Dokki, Giza, Egypt
(Email: mohamedraslan@eri.sci.eg)
(Received Jan. 15, 2017; revised and accepted May 16 & June 20, 2017)

Abstract

Public key cryptosystems depend on complex and time consuming arithmetic operations. Public key cryptosystems require modular operations over large numbers or finite fields. Researchers are working on improving the performance of asymmetric cryptosystems while maintaining the security of the cryptographic algorithms. Parallel computing is the most promising solution to improve the computing power and speed-up these arithmetic operations. In this paper, we propose a generic model to execute any encryption algorithm through a parallel-pipelined design. The proposed design is twofold. First, we make use of a number of processors to execute different encryption/decryption steps in parallel. Secondly, complex arithmetic operations could be divided into small simple arithmetic operations that are executed in parallel. Simulation experiments show that parallel implementations of the aggregated signcryption protocol (as a case study) outperforms the sequential performance. The average values of improvement ranges from 47.5% to 80.4% for different number of processors.

Keywords: Cryptography; Modular Operations; Parallelization; Public Key Cryptosystems

1 Introduction

Security is a serious issue when it comes to carry information over non-secure channels. In the era of Internet of Things, sensitive information requires secure transmission over the Internet. Cryptography is one of the most popular techniques that provide security and integrity to sensitive information over adversarial communication channels. Cryptographic algorithms are classified into two distinct categories: Symmetric and Asymmetric Key algorithms. Symmetric Key algorithms use a single key in order to encrypt, as well as, decrypt the data. Asymmetric key algorithms use two different keys in order to perform encryption and decryption. To carry out secure communication between contending parties, the receiver generates the public key and the corresponding private key. Then, the receiver shares the public key with the sender using a certificate authority. The sender encrypts the message using the public key and sends the cipher text to the receiver. The receiver decrypts the cipher text using corresponding private key and recovers the message. There are many public key based algorithms [28] (i.e. RSA, Digital Signature Algorithm, and Diffie-Hellman Key Exchange Algorithm.) Cryptographic algorithms are sequential algorithms. A single processor executes instructions one by one in sequence.

Asymmetric algorithms make use of modular arithmetic that requires complex computation for very large integers. Sequential implementation of complex asymmetric algorithms degrades the performance of encryption and decryption. Moreover, these algorithms demand huge memory size and have high power consumption. Parallelization techniques reduce the power consumption and achieve high performance in terms of execution time. Parallelization techniques use multi-core processors for efficient execution of instructions. In a parallel process, the processing is broken down into parts that are executed concurrently on different CPUs. Security algorithms can be implemented in parallel after dividing the algorithm into specific parts that can be executed on multi-core processor in order to enhance the performance.

Multimedia applications, such as, video on demand, distance learning, interactive e-Commerce, and TV channels delivery via Internet require secure communication over the Internet in order to transfer data. Cryptography protects data to achieve security services. Modern multimedia applications are real time applications that have concerns regarding delay introduced by cryptography. To expedite cryptographic computations, there are two approaches. First, design faster cryptographic algorithms. It is not a practical option, because standard cryptographic algorithms take time to be developed and tested. Moreover, performance of cryptographic algorithm depends on number of rounds or by the size of

the message. Secondly, to expedite cryptographic computations, we can deploy a parallel cryptographic system. The parallel encryption and signing was introduced by [9, 25, 36].

In this paper, we propose a generic model to execute any encryption algorithm (symmetric or public) through a parallel-pipelined design. We accelerate the cryptographic algorithms through parallel-pipelined design. We make use of *"M"* processing elements (PEs) to execute different encryption/decryption steps in parallel. Then, complex arithmetic operations could be divided into small simple arithmetic operations that are executed in parallel (load-balancing level). Simulation experiments show that parallel implementations of the aggregated signcryption protocol (as a case study) outperforms the sequential performance. The average values of improvement ranges from 47.5% to 80.4% for different number of processors.

This paper is structured as follows. Section 2 puts forward background and related work. Next, cryptographic arithmetic model is discussed in Section 3. The description of expedited asymmetric cryptosystems is presented in Section 4. Section 5 concludes the work.

2 Background and Related Work

Parallel techniques are used to accelerate several cryptographic algorithms (symmetric [10, 11, 12, 13, 29] and asymmetric [1, 3, 5, 7, 8, 18, 21, 22, 24, 27, 31, 33, 34, 35]). Practical asymmetric (public) cryptography requires modular operations over large numbers that is considered as computationally exhaustive process. Many researchers have done work in order to expedite the performance of asymmetric cryptography.

RSA is one of the most popular public key cryptography based algorithm. It is based on the mathematical scheme of factorization of very large integers which is a compute-intensive process and takes very long time. Several scientists used parallel computing to speed up the RSA algorithm. Saxena and Kapoor [31] presented a survey of various parallel implementations of RSA algorithm involving variety of hardware and software implementations.

Bajard and Laurent [1] presented a full implementation of RSA that has an efficient hardware implementation. It is sequential in nature but gives high throughput. Rawat and Walfish [27] presented a parallel signcryption standard using RSA with Probabilistic Signature and Encryption Padding (PSEP). Ciet *et al.* [7] presented a parallel FPGA implementation of RSA with Residue Number Systems (RNS). Tang *et al.* [33] presented a modular exponentiation technique using parallel multipliers. Wu *et al.* [34] offered a fast parallel technique has a speedup of 1.06 to 2.75. Liu *et al.* [22] proposed a high performance VLSI implementation of the RSA algorithm. Chang *et al.* [5] presented a fast parallel molecular algorithm for DNA-Based computation: factoring integers. Lin *et al.* [21] presented an efficient parallel RSA decryption algorithm for many-core Graphics Processing Unit (GPUs) with Compute Unified Device Architecture (CUDA.) Zhang *et al.* [35] presented a comparison and analysis of General-Purpose computing on Graphics Processing Units (GPGPU) and parallel computing on multicore CPU. Damrudi and Ithnin [8] presented a parallel RSA encryption based on tree architecture. Mahajan and Singh [24] presented an analysis of RSA algorithm using GPU programming. Mahajan and Singh described that the GPU as a coprocessor of CPU can be used to implement massive parallelism. Mahajan and Singh designed parallel RSA algorithm for GPU using CUDA framework and tested for both small and large prime numbers.

Elliptic Curve Cryptosystems (ECC) is used among the cryptographic community for their relatively better security and ease of implementation [3]. Several researchers used parallel computing to speed up the ECC algorithm. Basu [3] presented a parallel algorithm for elliptic curve cryptography (ECC). His simulation studies had been performed by implementing the parallel algorithm on a multi-core architecture (upto 8 cores). Hossain *et al.* [18] proposed a parallel architecture for fast hardware implementation of ECPM. It has been implemented over the binary field, and supported two Koblitz and random curves for the key sizes 233 and 163 bits.

In this work, we propose a generic parallel-pipelined design to speed up different encryption algorithms (symmetric or public). In the next section, cryptographic arithmetic model is discussed.

3 Cryptographic Arithmetic Model

Security services are generally classified into six components: confidentiality, data integrity, authentication, authorization, non-repudiation, and accountability. To achieve those issues different encryption/decryption protocols have been proposed, all those protocols are based on complicated mathematical operations. These operations can be divided into modulo operations (such as: modular addition, modular multiplication, modular subtraction, modular exponential, multiplicative inverse, and etc.), logical operations (such as: ANDing, inverse AND (NAND), ORing, XORing, Shift left, Shift right, Rotate left, Rotate right).

Some encryption techniques need more than one step (encryption or decryption) to complete the message encryption such as Aggregated Signcryption [4, 6, 15], Sign-Encrypt-Sign, Encrypt-Sign-Encrypt [26], and Schnorr Signcryption [30, 32]. The total execution/sequential time for one message (Tmess) can be calculated as follows:

$$Tmess = TKG + \sum_{l=1}^{\gamma} T_{step_l} \quad (1)$$

Where TKG is the key generation time, $Tstep$ is the time needed to execute encryption or decryption operation in

one step, and "γ" is the number of steps.

$$TKG = \sum_{l=1}^{key} T_{kg_l} \quad (2)$$

where, key is the number of keys and

$$Tstep = \sum_{i=1}^{K} T_{stagei} \quad (3)$$

where K is the number of the encryption/decryption stages.

To calculate the total time for any encryption or decryption stage T_{stagei}, we assume that this stage needs ai,1 modular additional operations, ai,2 modular subtraction operations, ai,3 modular multiplication operations, ai,4 modular exponential operations, ai,5 modular multiplication inverse operations, and ai,6 logical operations (may include: AND, OR, XOR, NAND, Rotate left, Rotate right and etc.). T_{stagei} is calculated as follows:

$$\begin{aligned} T_{stagei} = & \quad a_{1,i}t_{mod-add} + a_{2,i}t_{mod-sub} \\ + & \quad a_{3,i}t_{mod-mul} + a_{4,i}t_{mod-ex} \\ + & \quad a_{5,i}t_{mul-inv} + a_{6,i}t_{\log} \end{aligned} \quad (4)$$

Then,

$$Tstep = \sum_{i,j}^{K,f} a_{j,i}t_{operation(op)} \quad (5)$$

where operations $\in$\{Modular addition (mod-add), Modular subtraction (mod-sub), Modular Multiplication (mod-mul), Modular exponential (mod-ex), Modular multiplication inverse (mul-inv) and others logical operations\}, f = |Operation|, $1 \leq j \leq$ f, and $1 \leq i \leq$ K.

Figure 1 presented a general form of an encryption protocol. Assuming that, t_{add} is the time needed for computing simple addition operation, t_{sub} is the time needed for completing simple subtraction operation, t_{mul} is the time needed to execute simple multiplication operation, t_{div} is the time needed to perform simple division operation, and $t_{sub} = t_{add}$.

Modular addition can be calculated as the summation of addition and modulo operations. Using Barret algorithm [2], modulo operation needs one normal/simple multiplication, one normal division, and one normal subtraction. Then, the time needed to compute modular addition operation (which can be divided into one addition, one subtraction, one multiplication and one division), is $t_{mod-add}$ and is given by:

$$\begin{aligned} t_{mod-add} &= t_{add} + t_{modulo} \\ &= 2t_{add} + t_{div} + t_{mul} \end{aligned} \quad (6)$$

On the other hand, the time needed to compute modular subtraction operation which can be done by simple subtraction followed by modular addition; is $t_{mod-sub}$ and calculated as:

$$t_{mod-sub} = 3t_{add} + t_{div} + t_{mul} \quad (7)$$

Moreover, the time needed to compute modular multiplication operation, which can be divided into three simple multiplication operation and one simple addition operation [16] is $t_{mod-mul}$.

$$t_{mod-mul} = t_{add} + 3t_{mul} \quad (8)$$

Furthermore, to compute modular exponentiation operation using Indian algorithm [20], it needs approximately $\left(\frac{3b}{2}\right)$ modular multiplications for an exponent "e" of "b-bit", assuming the exponent contains approximately 50% ones/zeros. That is to say, a modular exponentiation "$Y = X^e mod n$" is performed by successive modular multiplication and the time needed to compute modular exponentiation operation t_{mod-ex} can be calculated as follows:

$$\begin{aligned} t_{mod-ex} &= \frac{3b}{2}t_{mod-mul} \\ &= \frac{3b}{2}(t_{add} + 3t_{mul}) \end{aligned} \quad (9)$$

On the other hand, modular multiplication inverse is executed using Euler's theorem [19], therefore, it could be done using modular exponential, and then the time needed to compute multiplication inverse operation can be computed as follows:

$$\begin{aligned} t_{mul-inv} &= t_{mod-ex} \\ &= \frac{3b}{2}(t_{add} + 3t_{mul}) \end{aligned} \quad (10)$$

From Equation (1) and Equation (4), the time needed to encrypt/decrypt this stream of data (N messages) on one processing element (Ts) is given by the following equation:

$$Ts = N * (\sum_{l=1}^{key} T_{kg_l} + \gamma \sum_{i,j}^{n,f} a_{j,i}t_{operation(op)}) \quad (11)$$

In case of all stages have the same operations; the time needed to encrypt/decrypt this stream of data on one processing element (PE) is given by:

$$Ts = N * (\sum_{l=1}^{key} T_{kg_l} + \gamma * \sum_{j}^{f} a_j t_{operation(op)}) \quad (12)$$

where $\sum_{j}^{f} a_j t_{operation}$ is time needed to execute any stage$_i$, $1 \leq i \leq$ K.

Due to the nature of most security algorithms, which are characterized by repeating the same function for several messages, different levels of parallelism can be used. This will improve the system utilization and throughput. In the next section, we use parallel and pipeline techniques to speed up these protocols.

4 Expediting Asymmetric Cryptosystems

Parallel systems, which emphasize parallel processing, are the most favorable architectures to increase the computing power and achieve speedup. Parallel processing continues to hold the promise of the solution of more complex problems, by connecting a number of powerful processing elements (PEs) together into a single system. These connected processors cooperate to solve a single problem that exceeds the ability of any one of the processing elements (PEs). That is to say, parallel computing is the simultaneous execution of the same task (split up) on multiple processing elements (PEs) in order to obtain faster results. The idea is based on the fact that the process of solving problem can be divided into small tasks, which

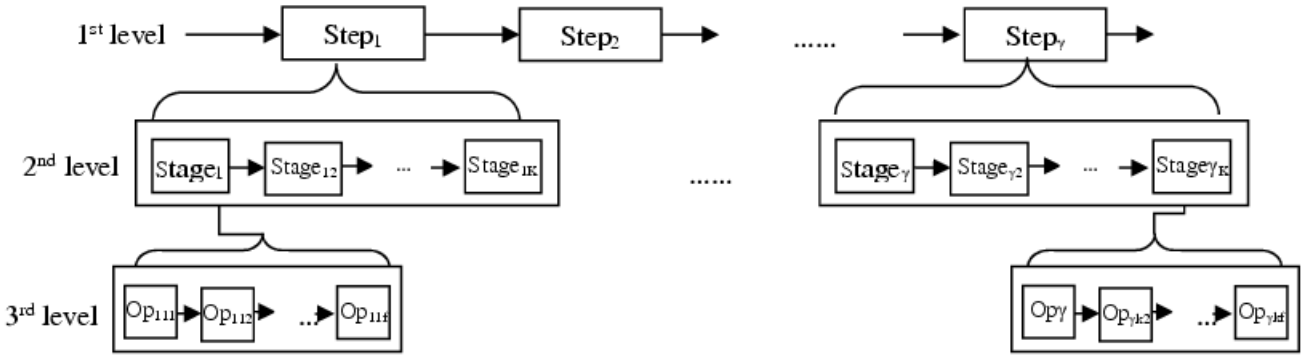


Figure 1: General form of an encryption/decryption protocol

may be carried out simultaneously with some coordination.

In our work, we will not concentrate on parallelizing the key generation process, since; it is done for several messages. However, we recommend making use of the parallel architecture in its implementation. From Equation (1), *Ts* for 'N' messages is calculated as follows:

$$TNs = N * \sum_{l=1}^{\gamma} T_{step_l} \quad (13)$$

Our proposed design is composed of three levels of parallelism: *first* different steps are pipelined, *then*, different stages inside each step are pipelined, and *finally*, load is balanced between PEs.

4.1 First Level: Pipelining Encryption/Decryption Processes

Pipelining is a kind of parallel computing that increases system performance by taking the advantage of the intrinsic parallelism by breaking a process into sub-processes executed by different PEs with inputs streaming through. There are two pipeline levels; the first one is pipelining different steps, while the second one is through pipelining the stages of each encryption/decryption process individually.

In this section, we will discuss how to pipeline different steps on a computer cluster equipped with 'γ' homogenous clusters 'C', where $C = \{C1,C2,..., C\gamma\}$. Assuming that the time needed to execute step$_l$ on cluster C_l is *Tstepl* the parallel/pipelined time *Tpar* is as follows:

$$T_{par} = N\ ^*T_{step_{max}} + (\sum_{l=1}^{\gamma} T_{step_l} - T_{step_{max}}) \quad (14)$$

where, $T_{step_{max}}$ is the needed time to execute the latest encryption/decryption operation. From Equations (13) and (14), the improvement in the execution time can be executed as follows:

$$\frac{T_{Ns}-T_{par}}{T_{Ns}} = \frac{(N-1)*(\sum_{l=1}^{\gamma} T_{step_l} - T_{step_{max}})}{N*\sum_{l=1}^{\gamma} T_{step_l}} \quad (15)$$

In case of repeated processes (all processes need the same execution time). Therefore, $T_{par}$is given by:

$$T_{par} = (N+\gamma-1)\ ^*T_{step_l} \quad (16)$$

From Equation (13) and Equation (16), the improvement in the execution time is executed as follows:

$$\frac{T_{Ns}-T_{par}}{T_{Ns}} = \frac{(N-1)*(\gamma-1)}{N*\gamma} \quad (17)$$

4.2 Second Level: Pipelining Different Stages of an Encryption/Decryption Operation

In this section, different stages of each step are executed in a stream of PEs in a pipelined manner separately. As mentioned at the previous section, each encryption/decryption process can be executed using a separate cluster, and assuming that inside each cluster there are 'M homogenous processing elements (PEs), where for cluster C_l, PE_l={$PE_{l,1}$,$PE_{l,2}$,..., $PE_{l,M}$}.That is to say inside each cluster 'M' PEs can cooperate to accelerate each process separately. To execute any encryption/decryption process using 'M' processing elements, there are three cases: i) number of PEs is smaller than the number of stages (M<K), ii) number of PEs equals the number of stages (M = K), and iii) number of PEs is greater than the number of stages (M > K).

4.2.1 Number of PEs (M) is Smaller than the Number of Stages (K)

In this case, to achieve the optimum system effectiveness one or more consecutive stages can be executed by each PE. In other words, the first PE_1 gives its output to the subsequent one PE_2. This is repeated for the different PEs. Assuming that the time needed to perform stage number 'i' is T_{stagei}, and from Equation (2), the total sequential time to execute 'N' message is:

$$T_S = N * \sum_{i=1}^{K} T_{stage_i} \quad (18)$$

Each PE_lneeds T_{PEl} to compute its assigned tasks, and the maximum time is:

$$T_{\max} = \max_{l=1}^{M} T_{PEl} \quad (19)$$

Then the parallel time *Tpar* is calculated as follows:

$$T_{par} = \sum_{i=1}^{K} T_{stage_i} + (N-1)T_{\max} \quad (20)$$

From Equation (18), and Equation (20), the improvement in the execution time can be calculated as follows:

$$\frac{T_s - T_{par}}{T_s} = \frac{(N-1)*(\sum_{i=1}^{K} T_{stage_i} - T_{\max})}{(N*\sum_{i=1}^{K} T_{stage_i})} \quad (21)$$

In case of repeated stages, the output of each stage$i$ is shifted to stage$i$+1 for all stages at the same time, and from Equation (13), the total sequential time can be calculated as follows:

$$Ts = N * K * T_{stage} \quad (22)$$

The parallel/pipelined time *Tpar* is calculated for the following two cases:

1) ***First***, M divides K (K/M= integer), where each PE executes the same number of stages, then the execution time *Tpar* is calculated as follows:

$$T_{Par} = (K + (N-1) * (\tfrac{K}{M})) * T_{stage} \quad (23)$$

The improvement in the execution time; which is achieved through using 'M' PEs, is given by:

$$\frac{T_s - T_{par}}{T_s} = \frac{(N*K) - ((N-1)*(\frac{K}{M}) + K)}{(N*K)} \quad (24)$$

2) ***Second,*** M does not divide K ($K/M \neq$ integer). For the first $(K - \lfloor K/M \rfloor \times M)$ PEs, each PE calculates$\lceil K/M \rceil$stages, and for the remaining PEs, each PE executes$\lfloor K/M \rfloor$stages. $T_{Par}$is given by:

$$T_{par} = (K + (N-1)*(|\tfrac{K}{M}| + 1)) * \mathrm{T}_{stage} \quad (25)$$

From Equations (18) and (25), the improvement in the execution time can be calculates as follows:

$$\frac{T_s - T_{par}}{T_s} = \frac{(N*K) - ((N-1)*(|\frac{K}{M}| + 1) + K)}{(N*K)} \quad (26)$$

4.2.2 Number of PEs (M) Equals to the Number of Stages (K)

In this case, the number of PEs equals to the number of stages to be executed (M=K). The total time to compute 'N' messages in parallel is given by the following equation:

$$T_{par} = (N-1) * T_{stage_{max}} + \sum_{i=1}^{K} T_{stage_i} \quad (27)$$

where $T_{stage_{max}}$ is the time needed to execute the latest stage, from Equations (3) and (27), the improvement in the execution time can be calculated as follows:

$$\frac{T_s - T_{par}}{T_s} = \frac{(N-1)*(\sum_{i=1}^{K} T_{stage_i} - T_{stage_{max}})}{N*\sum_{i=1}^{K} T_{stage_i}} \quad (28)$$

In case of repeated stages, the total time to compute 'N' messages in parallel is given by the following equation:

$$T_{par} = (N + K - 1) * T_{stage} \quad (29)$$

From Equations (18) and (29), the improvement in the execution time can be calculated as follows:

$$\frac{T_s - T_{par}}{T_s} = \frac{(N-1)*(K-1)}{(N*K)} \quad (30)$$

4.2.3 Number of PEs(M) is Greater Than the Number of Stages (K)

In this case, 'K' PEs are needed for computing 'K' stages and the remaining 'M-K' PEs are idle. This leads to load imbalance, to avoid load imbalance, more than one PE can work together to compute different operations. In other words, parallelism is accomplished in each stage's operation level as explained in the next section.

4.3 Third Level: Parallelization Inside Individual Stages (Load Balancing)

Parallelism is implemented in each stage's operation level. This is done in the operation/instruction level, where complicated operations could be reduced into simple operations in order to be executed in parallel. This combination will improve the system utilization and throughput. As assumed in Section 3.1, we simplify each stage into multiple operations as shown in Equation (4). Each encryption/decryption stage$_i$ needs *ai,1* modular additional operations, *ai,2* modular subtraction operations, *ai,3*modular multiplication operations, *ai,4* modular exponential operations, *ai,5* modular multiplication inverse operations, and *ai,6* logical operations.

The time to carry out logical operations is relatively very small compared to the modular operations. Hence, it will be neglected in our calculations. On the other hand, we divide the arithmetic operations into low time consuming arithmetic operations (modular additional, modular subtraction), and highly time consuming (modular exponential, modular multiplication, and modular multiplication inverse). For the low level time consuming operations there is one PE who executes the calculations. That is to say, it will be performed sequentially. While for the highly time consuming operations, we reduce all those operations into modular multiplications. As mentioned at Equation (9), modular exponential operation needs approximately (3b/2) modular multiplications. Similarly, modular inverse multiplication operation needs approximately (3b/2) modular multiplications as mentioned at Equation (10). Therefore, the efficient execution of the modular multiplication is the key to improve the performance. Then, our objective is to reduce the modular multiplication time. This can be achieved by incorporating the idle PEs in the computing of each modular multiplication operation.

Assuming that each stage has at least one modular multiplication, the modular multiplication operation can be divided into three simple multiplications and one addition (as mentioned before) and could be performed in parallel using three processing elements (PEs).

Case 1: $M = 3K$, two idle PEs can help each overloaded PEs to execute the modular multiplication operations, and the total parallel time can be decreased to approximately one-third of the time needed to execute the latest stage and calculated as follows:

$$T_{par} = \sum_{i=1}^{K} T_{stage_i} + (N-1)\tfrac{T_{\max}}{3} \quad (31)$$

Case 2: $M<3K$, the overloaded PEs must be arranged in an ascending manner, and the idle PEs must help the first $\left(\frac{M-K}{2}\right)$ overloaded PEs, and the total parallel time can be calculated as follows:

1) $\left(\frac{M-K}{2}\right) < 3$

$$T_{par} = \sum_{i=1}^{K} T_{stage_i} + (N-1)T_{ov} \qquad (32)$$

where 'T_{ov}': is the time needed by PE number $\left(\left|\frac{M-K}{2}\right| + 1\right)$(at the queue) to execute its task.

2) $\left(\frac{M-K}{2}\right) \geq 3$

$$T_{par} = \sum_{i=1}^{K} T_{stage_i} + (N-1)T_{ov} \qquad (33)$$

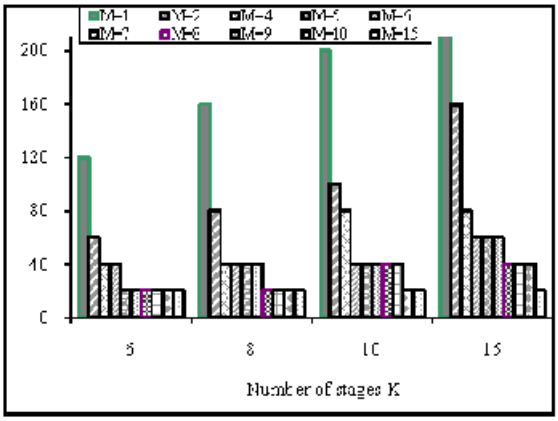
where 'T_{ov}': is the time needed by PE number $\left|\frac{M-K}{2}\right|$(at the queue) to execute its task.

4.4 Case Study: Aggregated Signcryption

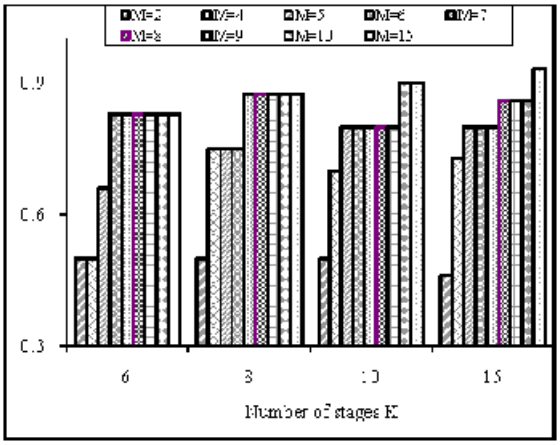
In computer security, digital certificates are verified using a chain of trust. The trust anchor for the digital certificate is the root certificate authority (CA). The certificate hierarchy is a structure of certificates that allows individuals to verify the validity of a certificate's issuer. Certificates are issued and signed by certificates that reside higher in the certificate hierarchy, hence the validity and trustworthiness of a given certificate is determined by the corresponding validity of the certificate that signed it. The chain of trust of a certificate chain is an ordered list of certificates, containing an end-user subscriber certificate and intermediate certificates (that represents the intermediate CA), that enables the receiver to verify that the sender and all intermediates certificates are trustworthy.

Aggregate signature scheme [4, 6, 15] is a digital signature that supports aggregation. It is a single short string that convinces any verifier that, for all $1 \leq i \leq N$, signer S_i signed message $Mess_i$, where the N signers and N messages may all be distinct. The main motivation of aggregate signatures is compactness. That is to say, aggregate signatures are useful for reducing the size of certificate chains (by aggregating all signatures in the chain) and for reducing message size in secure routing protocols. In addition, aggregate signatures perform verifiably encrypted signatures. Such signatures enable the verifier to test that a given ciphertext C is the encryption of a signature on a given message '*Mess*' [4]. Bonds *et al.* [4] introduced the concept of an aggregate signature which is based on bilinear pairing, and gave several applications for aggregate signatures. In case of signers is ordered, the aggregate signature is computed by having each signer, in turn, add his signature to it [23]. In this section, we implement our proposed design on this scheme.

$$e(\sigma, g_2) = e(\prod_i h_i, v_i) = \prod_i e(h_i, v_i) \qquad (34)$$



(a) Parallel execution time



(b) Improvement degree

Figure 2: The system performance when a member of messages $N = 20$, for different values of $K = 6, 8, 10$, and 15

Here '$\gamma = 3$' and from Equation (1), Ts for 'N' messages is calculated as follows:

$$T_{Ns} = \sum_{l=1}^{3} T_{step_l} \qquad (35)$$

In order to simplify the calculations, we will neglect the time required to calculate Steps 2 and 3 as they are small compared to the time required to perform bilinear mapping operation (Step 1), In addition, we assume that Step 1 is done in 'K' stages, and all stages have the same execution time regardless the key or message length. From Equation (22) the total sequential time can be calculated as:

$$Ts = N * K * T_{stage} \qquad (36)$$

In the following paragraphs, we illustrate the three cases performing the aggregated signcryption.

Case 1: Distinct messages (N) with multiple signers $M < K$. There are three cases:

1) K/M = integer, the parallel execution time and its improvement over sequential time is given by

:

$$T_{Par} = (N * (\frac{K}{M})) * T_{stage} \quad (37)$$

$$\frac{T_s - T_{par}}{T_s} = \frac{(N*K)-(N*(\frac{K}{M}))}{(N*K)} \quad (38)$$

2) $K/M \neq$ integer, the parallel execution time and its improvement over sequential time is given by :

$$T_{par} = (N*(|\frac{K}{M}| + 1))* \mathrm{T}_{stage} \quad (39)$$

$$\frac{T_s - T_{par}}{T_s} = \frac{(N*K)-(N*(|\frac{K}{M}|+1))}{(N*K)} \quad (40)$$

3) M=K, the execution time and its improvement over the sequential time can be calculated as shown:

$$T_{Par} = N * T_{stage} \quad (41)$$

$$\frac{T_s - T_{par}}{T_s} = \frac{(N*M)-(N)}{N*M} = \frac{M-1}{M} \quad (42)$$

Figure 2 presents the performance of the proposed model compared to the performance prior to parallelization for N = 20, and 'M ' ranges from 1 to 10 for different number of signers 'K' = 6,8,10, and 15 respectively.

1) Figure (2-a) demonstrates the total execution time (parallel time in terms of T_{stage}) for different values of 'K'. For different values of 'K', as the number of PEs increases, the total execution time decreases irrespective of the value of 'K'. When the nu mber of PEs is increased than ($K/2$), the execution time will be stabilized until the number of PEs reaches the number of signers 'K'. When M=K, a significant decrease in time occurs. Increasing the number of PEs than the number of stages 'K' leads to load imbalance. Consequently, the system's efficiency will decrease. Therefore, the number of PEs must not exceed a certain number which is called system's saturation. That is to say the saturation occurs when the number of PEs equals 'K'.

2) Figure (2-b) illustrates the degree of improvement of the proposed model compared to prior to parallelization. For different values of 'K', such as previously observed (execution time), as the number of PEs increases, the improvement degree increases irrespective of the value of 'K'. When the number of PEs is increased than ($K/2$), the improvement degree will be saturated until the number of PEs reaches the number of signers 'K'. When M=K, a considerable improvement occurs. For different values of 'K', the average values of improvement are 50%, 66.6%, 75.5%, 79.9% and 85.4% for M= 2, 3, 4, 6 and 8 respectively.

Case 2: Distinct messages (N) with ordered signatures.

In this case, different stages of Step 1 are executed in a stream of PEs in a pipelined manner separately. As mentioned at the previous section. We assumed repeated stages and we have two cases:

1) ***M<K***, and as mentioned above there are two cases:
   a. K/M= integer, the execution time and its improvement over the sequential is given by Equations (23) - (24).
   b. $K/M \neq$ integer the execution time and its improvement over the sequential is given by Equations (25) - (26).
2) M=K, execution time and its improvement over the sequential is given by Equations (29) - (30).

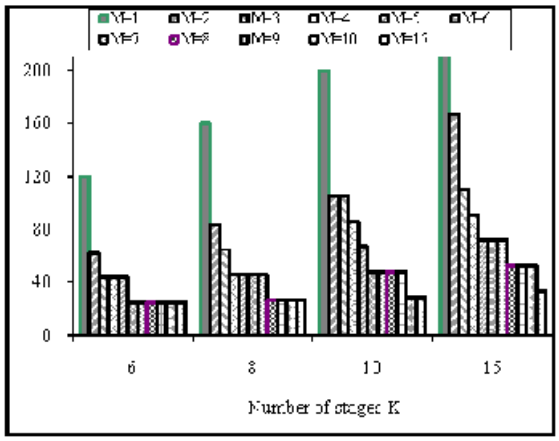
Figure 3 shows the performance of the proposed model with respect to the performance prior to pipelining for N = 20, and M = 1,2,3,4,5,6,7,8,9 and 10 for different number of signers 'K' = 6,8,10, and 15 correspondingly.

Figure (3-a) summarizes the total parallel time (in terms of T_{stage}) for different values of 'K'. As the number of PEs increases, the total execution time decreases irrespective of the value of 'K'. When the number of PEs is increased than ($K/2$), the execution time will be stabilized until the number of PEs reaches the number of signers 'K'. When M=K, a significant decrease in time occurs. When the number of PEs is increased than 'K', load imbalance will occur. Consequently, the system's efficiency will decrease. Therefore, the number of PEs must not exceed a certain number which is called system's saturation. That is to say the saturation occurs when the number of PEs equals 'K'.

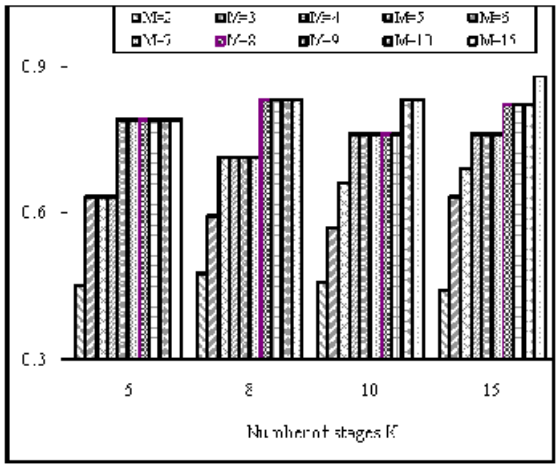
Figure (3-b) illustrates the degree of improvement of the proposed model compared to prior to parallelization. For different values of 'K', as the number of PEs increases, the improvement degree increases irrespective of the value of 'K'. Such as previously observed (execution time), when the number of PEs is increased than ($K/2$), the improvement degree will be saturated until the number of PEs reaches the number of signers 'K'. When M=K, a considerable improvement occurs. For different values of 'K', the average values of improvement are 47.5%, 60.6%, 67.3%, 71.5% and 80.4% for M= 2, 3, 4, 6 and 8 respectively.

From Figures 2 and 3, we can deduce that as increasing the number of 'K', the execution time increases. This is due to the fact that the computation time increases as increasing the number of signers. This shows the advantage of using a multiprocessor system in enhancing the system performance.

Case 3: Single message with distinct signers.



(a) Parallel execution time



(b) Improvement degree

Figure 3: The performance of the proposed model with respect to the performance prior to pipelining for $N = 20$, and $M = 1, 2, \cdots, 10$ for different number of signers $K = 6, 8, 10$, and 15

One or more PEs co-operates to executee $(h, \prod_i v_i)$. In this case the parallelization is done on the level of point addition operation. Point addition operation requires the execution of many Montgomery multiplications which consume time. Several techniques are proposed to parallelizing Montgomery multiplications [14, 17]. In this work, we will not concentrate on performing this case. In future work, we will investigate using the proposed architecture to execute Montgomery multiplication to solve the problem of load imbalance.

5 Conclusions

Security services are generally classified into six components: confidentiality, data integrity, authentication, authorization, non-repudiation, and accountability. Cryptography is one of the most popular techniques that can be used to provide sufficient security to the sensitive data. To achieve those issues different protocols have been proposed, all those protocols are based on complicated mathematical operations which are time consuming. Parallel systems, are the most favorable architectures to increase the computing power and expedite these operations. Parallel processing continues to hold the promise of the solution of more complex problems, by connecting a number of powerful processors together into a single system. These connected processors cooperate to solve a single problem that exceeds the ability of stand alone processor. In this work, we propose a generic model to execute any encryption algorithm (symmetric or public) through a parallel-pipelined design.

We address the problem of expediting the public key cryptographic algorithms by using parallel-pipelined design. Therefore, the total computation time is reduced. Parallel/pipelined technique reduces the computation time required to execute the modular multiplication operation, compared to its corresponding values of sequential execution in order to achieve high performance and throughput in public key cryptography. The proposed design makes use of 'M' processing elements to execute different encryption/decryption phases in parallel. Furthermore, it implements the parallelization mechanism on the arithmetic operation level (load-balancing level), where complex arithmetic operations could be divided into small simple arithmetic operations that are executed in parallel. Simulation experiments show that parallel implementations of the aggregated signcryption protocol (as a case study) outperforms the sequential performance (for different values of 'K') for both distinct messages with multiple signers and distinct messages with ordered signatures cases. For the first case the average values of improvement are 50%, 66.6%, 75.5%, 79.9% and 85.4% for M= 2, 3, 4, 6 and 8 respectively. While for distinct messages with ordered signatures case the average values of improvement are 47.5%, 60.6%, 67.3%, 71.5% and 80.4% for M= 2, 3, 4, 6 and 8 respectively.

Although, many researchers have done work in order to speed up the performance of cryptosystems using parallel computing, these algorithms are protocol specific. In contrast, we propose a generic model to execute any encryption algorithm through a parallel-pipelined design.

References

[1] J. Bajard and I. Laurent, "A full RNS implementation of RSA," *IEEE Transactions on Computers*, vol. 53, no. 6, pp. 769-774, 2004.

[2] P. Barret, "Implementing the Rivest Shamir and Adleman public key encryption algorithm on standard digital signal processor," in *Advances in Cryptolgy (Crypto'86)*, LNCS 263, pp. 311-323, Springer-Verlag, 1987.

[3] S. Basu, "A new parallel window-based implementation of the Elliptic Curve point multiplication in multi-core architectures," *International Journal of Network Security*, vol.14, no.2, pp.101-108, Mar. 2012.

[4] D. Boneh, D. Gentry, B. Lynn, and H. Shacham, "Aggregate and verifiably encrypted signatures from bilinear maps," in *Proceedings of Advances in Cryptog-raphy (EUROCRYPT'03)*, LNCS 2656, pp. 416-432, Springer-Verlag, 2003.

[5] W. L. Chang, M. Guo, and M. S. Ho, "Fast parallel molecular algorithms for DNA-based computation: factoring integers," *IEEE Transactions on Nano Bioscience*, vol. 4, no. 2, pp. 149-163, 2005.

[6] C. C. Chen, H. Chien, and G. Horng, "Cryptanalysis of a compact certificateless aggregate signature scheme," *International Journal of Network Security*, vol.18, no.4, pp.793-797, July 2016

[7] M. Ciet, N. Michael, P. Eric, and J. J. Quisquater, "Parallel FPGA implementation of RSA with residue number systems-can side-channel threats be avoided?," in *Proceedings of 46th Midwest IEEE Symposium on Circuits and Systems*, vol. 2, pp. 806-810, 2003.

[8] M. Damrudi and N. Ithnin, "Parallel RSA encryption based on tree architecture," *Journal of the Chinese Institute of Engineers*, vol. 36, no. 5, pp. 658-666, 2012.

[9] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644-654, 1976.

[10] V. Digraskar, A. Chirde, A. Jagtap, and A. Deasi, "Parallel computation of advance encryption standard algorithm for performance improvement," in *Proceedings of International Conference on Recent Trends in Engineering Science and Technology (ICRTEST'17)*, pp. 139–142, Jan. 2017.

[11] V. Digraskar, A. Chirde, A. Jagtap, and A. Deasi, "Secure file transmission using parallel Aes algorithm," *International Journal of Innovative Research in Computer and Communication Engineering*, vol. 5, no.3, Mar. 2017.

[12] C. L. Duta, G. Michiu, S. Stoica, and L. Gheorghe, "Accelerating encryption algorithms using parallelism," in *Proceedings of 19th International Conference on Control Systems and Computer Science*, May 2013.

[13] G. F. El Kabbany, H. K. Aslan, and M. N. Rasslan, "A design of a fast parallel-pipelined implementation of AES: Advanced Encryption Standard," *International Journal of Computer Science & Information Technology*, vol. 6, no. 6, pp: 39-45, Dec. 2014.

[14] J. Fan, K. Sakiyama, and I. Verbauwhede, "Elliptic curve cryptography on embedded multicore systems," *Design Automation for Embedded Systems*, vol. 12, no. 3, pp. 231–242, 2008.

[15] C. Gentry and Z. Ramzan, "Identity based aggregate signature," in *Proceedings of International Workshop on Public Key Cryptography (PKC'06)*, pp 257-273, 2006.

[16] J. GroBschadl, "High-speed RSA hardware based on Barret's modular reduction method," in *International Workshop on Cryptographic Hardware and Embedded Systems (CHES'00)*, LNCS 1965, pp. 191–203, Springer-Verlag, 2000.

[17] N. Guillermin, "A high speed coprocessor for elliptic curve scalar multiplications over FP," in *Proceedings of 12th International Conference on Cryptographic Hardware and Embedded Systems (CHES'10)*, pp. 48-64, 2010.

[18] M. Hossain, E. Saeedi, and Y. Kong , "Point-multiplication architecture using combined group operations for high-speed cryptographic applications," *PLoSOne*, vol. 12, no. 2, May 2017.

[19] K. Ireland and M. Rosen, *A Classical Introduction to Modern Number Theory*, 2ed., Springer-Verlag, 1990.

[20] D. Knuth, *Semi Numerical Algorithms*, The Art of Computer Programming, vol. 2, Addison-Wesley, Reading, MA, USA, 1969.

[21] Y. Lin, C. Lin, and D. Lou, "Efficient parallel RSA decryption algorithm for many-core GPUs with CUDA," in *Proceedings of International Conference on Telecommunication Systems, Modelling and Analysis (ICTSM'12)*, pp. 85-94, 2012.

[22] Q. Liu, F. Ma, D. Tong, and X. Cheng, "A Regular Parallel RSA Processor," in *Proceedings of 47th Midwest Symposium on Circuits and Systems (MWSCAS'04)*, vol. 3, pp. III-467-70, 2004.

[23] A. Lysyanskaya, S. Micali, L. Reyzin, and H. Shacham, "Sequential aggregate signatures from trapdoor permutations," in *International Conference on the Theory and Applications of Cryptographic Techniques, (EUROCRYPT'04)*, pp. 74-90, 2004.

[24] S. Mahajan and M. Singh, "Analysis of RSA algorithm using GPU programming," *International Journal of Network Security & Its Applications*, vol. 6, no.4, 2014.

[25] A. Menezes, S. Vanstone, and P. Oorschot, *Handbook of Applied Cryptography*, CRC Press, Inc., Boca Raton, FL, USA, 1996.

[26] M. Rasslan and H. Aslan, "On the security of two improved authenticated encryption schemes," *International Journal of Security and Networks*, vol. 8, no. 4, pp. 194-199, 2013.

[27] A. Rawat and S. Walfish, *A Parallel Signcryption Standard Using RSA with PSEP*, Technical Report, 2003.

[28] R. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120-126, 1978.

[29] V. H. Sathawane and T. Diwan, "An optimization parallel computation of advance encryption algorithm using Open-MP- a review," *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 7, no. 2, pp. 384-386, Feb. 2016.

[30] L. Savu, "Signcryption scheme based on Schnorr digital signature," *Journal of Software Engineering and Applications*, vol. 05, no. 02, Jan. 2012.

[31] S. Saxena and B. Kapoor, "State of the art parallel approaches for RSA public key based cryptosystems," *International Journal on Computational Sciences & Applications*, vol.5, no.1, Feb. 2015.

[32] C. P. Schnorr, "Efficient identification and signatures for smart cards," in *Advances in Cryptology (Crypto'89)*, LNCS 435, pp. 239-252, Springer-Verlag, 1990.

[33] S. Tang, K. Tsui, and P. Leong, "Modular exponentiation using parallel multipliers," in *Proceedings of IEEE International Conference on Field-Programmable Technology*, pp. 52-59, Dec. 2003.

[34] C. Wu, D. Lou, J. Lai, and T. Chang, "Fast parallel exponentiation algorithm for RSA public-key cryptosystem," *Informatica*, vol. 17, no. 3, pp. 445-462, 2006.

[35] D. Zhang, H. Hang, and X. Bi, "Comparison and analysis of GPGPU and parallel computing on multicore CPU," *International Journal of Information and Education Technology*, vol. 2, no. 2, pp.185-187, 2012.

[36] Y. Zheng, "Digital signcryption or how to achieve cost (signature & encryption) cost (signature) + cost (encryption)," in *Proceedings of the International Conference of CRYPTO'97*, LNCS 1294, pp. 165-179, Springer-Verlag, 1997.

Biography

Mohamed 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 Ph.D. from Concordia University, Canada 2010. His research interests include Cryptology, Digital Forensics, and Networks Security.

Ghada El-kabbany 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. Her research interests include High Performance Computing (HPC), Image Processing, Computer Network Security and Digital Forensics.

Heba Kamal Aslan is a Professor at Electronics Research Institute, Cairo-Egypt. She received her B. Sc. degree, M. Sc. degree, and Ph. D. degree from Faculty of Engineering, Cairo University, Egypt in 1990, 1994 and 1998 respectively. Her research interests include Key Distribution Protocols, Authentication Protocols, Logical Analysis of Protocols and Intrusion Detection Systems.