PACMA : An Adaptive Symmetric Cryptographic Algorithm for Parallel Computing Environments

J. John Raybin Jose a *, E. George Dharma Prakash Raj b

a * Bishop Heber College (Autonomous), Tiruchirappalli - 620017, India
b Bharathidasan University, Khajamalai Campus, Tiruchirappalli – 620023, India

Abstract

Traditional symmetric cryptographic algorithms are designed considering the cryptographic complexities rather than the processing capacities of the modern computing systems. Hence the performances of these algorithms are inconsistent and vary enormously when implemented in different computing systems with different processing capabilities. Parallelized adaptive cryptographic algorithms that can fit into the nature of the processing devices are the need of the hour. “Parallelized Adaptive Cipher with Modular Arithmetic” (PACMA) is a different class of cryptographic algorithm involving modular arithmetic concepts, which addresses these issues effectively. The size of the key and the plain text blocks are each 1024-bits. This symmetric block cipher is made adaptive by incorporating flexibility in the size of the key and plain text sub-blocks and the number of rounds. Level of Intra-packet parallelization, variety in grain size and the required security strength are achieved by suitably deciding the sub-block size. Flow of the algorithm is made dynamic by determining the execution steps through each key value at the runtime. The performance of the PACMA is analyzed with implementations in shared memory parallel programming environment using OpenMP, Java Threads and MPI.

Keywords: Symmetric Block Cipher, Adaptive Cryptographic Algorithm, Modular Arithmetic, Intra-packet Parallelization, Sub-block Size, Speedup.

1. Introduction

Cryptography is all about the art and science of hiding Information. Cryptographic techniques are very much helpful in preserving the confidentiality, integrity, authenticity and validity of information in move or in store. Traditional view on the efficiency of a cryptographic algorithm focuses only on the complexity of the algorithm and the strength and the secrecy of the key [1]. Wide varieties of challenges are faced in efficiently implementing these crypto systems, as new trends and technologies have crept into modern communication and computing systems.

Conventional symmetric cryptographic algorithms are developed before the year 2000 when computers were built around 32, 16 or 8 bit processors. But now, Cryptographic algorithms can be executed much faster on modern computers. The Computers of today, and most definitely that of tomorrow, is not that of 32-bit desktops, rather one of multi-cored chips and multiprocessor machines whose processing capacity is 64 or 128 or more bits. Parallelizing the cryptographic algorithms is a means to utilize these systems effectively [2]. Moreover, the rate of encryptions and decryptions carried out per unit time in communication systems has increased exponentially. This imposes additional overhead in the information exchange and causes congestion. A way out of this trouble is to develop a new class of parallel algorithm that can reduce the time required for encryption and decryption using parallel execution techniques, without compromising the security strength.

This paper is designed as follows. Section 2 gives the Cryptographic Techniques used, Section 3 highlights the Related Works, description of Parallel Adaptive Cipher with Modular Arithmetic (PACMA) is given in section 4, Implementation and Analysis is dealt in section 5 and Section 6 concludes the paper.
2. Cryptographic Techniques

Cryptographic algorithms are classified into Symmetric and Asymmetric Key Algorithms. Symmetric key algorithms use the same key for encryption and decryption and so they rely on the secure distribution and management of the session key. Symmetric key algorithms can be divided into stream ciphers and block ciphers. Stream ciphers encrypt the bytes of the message one at a time but block ciphers take a number of bytes and encrypt them as a single unit. In Asymmetric Key Algorithms a public key and a private key are used. The public key is used to encrypt the information at the sending end and it is available to all; whereas the private key is known only to the receiver and it is used to decrypt the information. The issue of key distribution is overcome in asymmetric cryptography [3, 4].

Modular Arithmetic revolves around the remainder of division operation. It is represented by the relation: \( x \mod m = r \). Here ‘x’ is an integer from the set of integers ‘Z’. A positive integer ‘m’ known as modulus is used to divide the value ‘x’ to produce a positive integer known as residue ‘r’ from the set of residues \( Z_m \). The set of residues comprises all the elements between 0 and \( m-1 \). Modular arithmetic allows the binary operations; addition, subtraction and multiplication to be applied on the elements of \( Z_m \). After applying each operation, the result obtained is mapped to \( Z_m \) with the help of the modulus operator. There exists many-to-one relationship between the elements of ‘Z’ and \( Z_m \) and this relationship is known as congruence.

Reversible operations are involved in cryptography to achieve encryption and decryption. Addition and multiplication modulo operations are reversible and can be efficiently involved in cryptography. The binary coded value of the plain text is added or multiplied with the key, which is a member of the set of residues \( Z_m \) to yield a sum or product value. When the value obtained is divided with ‘m’ yields a remainder, which is the resultant cipher text. Similarly the plain text can be retrieved by performing the same operation with the cipher text and the inverse value of the key used. Therefore it is necessary to find the additive or multiplicative inverse of an element with respect to the operation used for decryption. Each element in modular arithmetic has only one additive inverse, which is always unique and at sometimes, the additive inverse of an element is the element itself. If ‘x’ and ‘y’ are two elements of the set \( Z_m \), ‘x’ is said to be the additive inverse of ‘y’ and vice versa if: \( x + y \equiv 0 \mod m \).

An element may or may not have a multiplicative inverse. If ‘x’ and ‘y’ are the two elements of the set \( Z_m \), then ‘x’ is said to be the multiplicative inverse of ‘y’ and vice versa if \( x \cdot y \equiv 1 \mod m \). The simple method to determine whether or not a number ‘x’ in \( Z_m \) has a multiplicative inverse is to compute the GCD of ‘x’ and ‘m’. If gcd(x, m) = 1 then ‘x’ and ‘m’ are said to be relatively prime and ‘x’ has a multiplicative inverse; otherwise, the multiplicative inverse for ‘x’ in \( Z_m \) does not exist and these values of \( Z_m \) cannot be used for the multiplicative modulo operations [3-6].

3. Related Works

Parallelism accelerates processing by simultaneous execution of multiple tasks. Implicit parallelism is achieved by the inherent resources and techniques in the processing hardware. Explicit parallelism is extracted by the external arrangements and codes by utilizing the available parallel hardware resources efficiently. Techniques used for explicit parallelism can be categorized as (i) Per-Connection Parallelism (ii) Per-Packet Parallelism and (iii) Intra-Packet Parallelism.

Per-connection parallelism is a method in which each connection is given its own thread or process that runs exclusively on one processor. This is the most common method of parallelization, and requires no modification to the existing algorithm and often no modification to the existing software. The per-connection parallelization method makes no attempt to fully utilize modern architectures. In Per-packet parallelism connections disperse their packet processing load over multiple processors, wherein each packet is treated individually. Many current algorithms lend themselves well to this kind of parallelization, but, surprisingly, no cryptographic software implementing this per-packet parallelism is available. Intra-packet parallelism is the most difficult type of parallelism, as it depends on algorithm design. It also requires changes in the implementation of the cryptographic algorithm, depending no longer on the flexibility of the hardware or operating system upon which it is run [2, 7, 8, 9]. Intra-packet parallelism is employed in PACMA with the ability to adjust itself according to parallel environment in which it is executed.

Efforts to parallelize existing cryptographic algorithms have been pursued by several researchers from year 2000 onward. The prominent of these efforts can be classified broadly as Hardware or Software Parallel Cryptographic Implementations involving several technical approaches beneath them as depicted in fig 1.

![Parallel Cryptographic Implementations](image)

**Fig 1: Classification of Parallel Implementation of Cryptographic Algorithms**

Design and implementation of a crypto processor by HoWon Kim et al, 2002, introduced a special-purpose
microprocessor optimized for the execution of cryptography algorithms. The crypto processor consists of a 32-bit RISC processor block and a coprocessor block dedicated to the SEED and Triple-DES algorithms [10]. Pionteck et al. 2004, in their work presented a hardware design of AES with reconfigurable encryption/decryption engines which supports all key lengths. The reconfigurable crypt-engine is integrated in a 32 bit RISC processor as a functional unit and can operate in parallel with the standard ALU. The Reconfigurable Cryptographic Unit (RCU) is integrated into a 32 bit five stage pipeline RISC Processor and the area which is used for the RCU is less than 27% of the total area [11].

An Application-Specific Integrated Circuit (ASIC) is an Integrated Circuit (IC) customized for a particular use, rather than intended for general-purpose use. When ASICs are used to implement cryptographic algorithms it provides robust operation and much of the overhead involved in hardware implementation is reduced. The works carried out on ASIC implementation of DES, 3DES, IDEA and all the round 2 candidates of AES by S. Mukherjee et al., 2011, B. Weeks et al., 2000 and T. Ichikawa et al., 2000 are prominent in this category [12-14].

Field Programmable Gate Array (FPGA) logic cells are reconfigurable platforms that provide relatively a low cost, high performance method of implementing cryptographic primitives. Several standard algorithms such as DES, Triple DES, and AES are parallelized using FPGAs by Swankoski et al. [15]. The implementation environment is composed of Virtex-II Pro, FPGA Platform with Verilog HDL and Block RAM resources. Another hardware design of AES in chip proposed by Kotturi, et al., uses hierarchical simultaneous key generation, is implemented with ten separate units in XC2VP70 device with speed grade -7 with Virtex II Pro FPGA. Each unit can execute one round of the algorithm and ten rounds of the algorithm can be executed in parallel in a chip using external pipelined design. The throughput rate achieved in this method is higher than most other implementations [16]. In yet another implementation by Chi-Wu, et al, 128-bit AES was decomposed into four 32-bit AES to be executed in parallel. This outperformed all other recent works by requiring less than 20% reconfigurable area and operated four times faster than 32-bit AES by providing double the throughput [17].

Microprocessors with multiple cores and Graphical Processing Units (GPUs) are widely available at affordable prices. Considering the computational demands of the cryptographic algorithms, these parallel platforms are relevant to parallelize the existing algorithms to enhance the performance. CUDA programming is used to parallelize the algorithms in GPU [18]-[22]. OpenMP is used to extract parallelism from Multi-Core Processors [23].

Considering the fact that the most time-consuming elements of source code of cryptographic algorithms without including the I/O functions are loops, they are parallelized for all the popular cryptographic algorithms such as DES, Triple DES, IDEA, AES, RC5, Blowfish, GOST and LOK191 by Burak et al. The standard modes of operations selected for Ciphers are ECB, CBC, CFB, OFB and CTR. The Data Dependences are resolved before parallelizing the Loops. OpenMP was used to parallelize the loops in these algorithms and Petit was used to resolve dependences in the loops. Speedup measurements were presented for all these implementations [23-28].

Even though all the efforts to parallelize the existing conventional cryptographic algorithms with hardware and software techniques had given better results, they cannot be fully parallelized or implemented efficiently in present day computing systems. The dependency problems and the inability to efficiently modularize the sections of the algorithms hover around and haunt the parallelization. Thus a path for the new class of cryptographic algorithms that is devoid of these problems is set in.

4. Parallel Adaptive Cipher With Modular Arithmetic (PACMA)

PACMA is a symmetric block cipher with the block length and the key size each of 1024 bits. The sub-block size of the key and the plain text is made adaptive by varying them suitably based on the processing capacity of the computing system used. The behavior of the algorithm is decided dynamically by deriving the control information from the key. The granularity of the algorithm is decided by forming sub-blocks of various sizes in the range 2^n where n=3 to 8. The processing resources available and the security strength required are used to decide the number of rounds, size of the key and the plaintext sub-blocks. This is depicted in fig 2.

![Fig 2. General Block Diagram of PACMA](image)

Each round in PACMA has eight stages as depicted in fig 3. The 1024 bit key is directly used in the first stage but it undergoes different transformations in the remaining stages. The sub-block generation routine is run to generate the key and the plain text sub-blocks; before they are involved in operations at each stage. In the first stage modulo addition operation is performed with the key sub-blocks and the plain text sub-blocks in a pattern decided by the initial and the final bits of the key. If both these bits are of same value
then modulo addition operation is performed directly, otherwise the plain text bits are reversed before the operation. The key bits are then rotated to right or left by \( b/2 \) positions within the key sub-blocks so that it can be used in the next stage. Here \( b \) refers to the number of bits in each key sub-block. The direction of rotation is determined by the parity of the key. If the parity is odd the bits are rotated to the right, otherwise they are rotated to the left.

In the second stage addition modulo \( 2^8 \) operation is carried out with each key and plain text sub-blocks. If the sub-block size is greater than 8 bits, the key and the plain text sub-blocks are further divided into chunks of 8 bits in this stage to facilitate the addition modulo \( 2^8 \) operation. Following this exchange manipulation is carried out between the key sub-blocks. This is achieved by swapping the odd numbered sub-blocks with the next higher order even numbered sub-blocks. In the third stage, multiplication modulo \( 2^8+1 \) operation is carried out on the 8 bit chunks of the key and the plain text that are available after the second stage. Intra sub-block rotation is then carried out for each 8 bit chunk of key sub-blocks as it was done before the second stage.

In the fourth stage addition modulo \( 2^{16} \) operation is performed with the key sub-blocks and the plain text sub-blocks after dividing or grouping them into chunks of 16 bits. An inter sub-block rotation is performed for the key done before the fifth stage. The position for rotation is derived from the value given by the first ten bits of the key block. The direction of rotation is specified by the eleventh bit of the key block. If the value of the eleventh bit is ‘1’ the key bits are rotated to the right and if it is ‘0’ then it is rotated to the left. Multiplication modulo \( 2^{16}+1 \) operation is carried out after further dividing or grouping the key and plain text sub-blocks into chunks of 16 bits in the fifth stage. The key manipulation is then performed with the 16 bit chunks of the key as it is done before the fourth stage for 8 bits chunks.

The operation in the second state is repeated for the sixth stage. The key sub-blocks then undergo an exchange key manipulation by swapping the sub-blocks at the extreme ends. The inner sub-blocks at the next level are then swapped and this is carried towards the sub-blocks in the central position of the key. The operation performed in the third stage is repeated in the seventh stage. The key bits then experience the Intra sub-block rotation similar to the first key manipulation operation and then they are utilized for modulo addition operation in the eighth stage. Brief algorithmic depiction of PACMA with single round can be given as follows:

**Input:** 1024 bit plain text block

**Output:** 1024 bit cipher text block

**Sub-Block Generation:**

1. Run environment identification routine to identify the number of processors/cores ‘\( p \)’, their data handling capacities ‘\( c \)’ and clock speed ‘\( s \)’ to divide the 1024 bits key and the 1024 bits plain text into sub-blocks of ‘\( b \)’ bits.

Fig 3. Stages in each round of PACMA

2. if \( p==1 \) && \( c < 16 \) bits && \( s \leq 10 \) MHz then \( b=8 \) bits.
3. else if \( p==1 \) && \( c \geq 16 \) bits && \( c < 32 \) bits && \( s > 10 \) MHz && \( s \leq 100 \) MHz then \( b=16 \) bits.
4. else if \( p==1 \) && \( c \geq 32 \) bits && \( c < 64 \) bits && \( s > 100 \) MHz && \( s \leq 1000 \) MHz then \( b=32 \) bits.
5. else if \( p \leq 4 \) && \( c \geq 64 \) bits && \( s > 1 \)GHz then \( b=64 \) bits.
6. else if \( p > 10 \) c \( \geq 64 \) bits && \( s > 3 \)GHz then \( b=256 \) bits
7. else display “resources unsuitable to implement PACMA”

**Steps in Single Round of PACMA:**

1. Modulo addition operation between key and plaintext sub-blocks.
2. Intra sub-block rotation on key sub-blocks.
3. Addition modulo \( 2^8 \) operation between key and plaintext sub-blocks.
4. Exchange sub-block operation on key sub-blocks.
5. Multiplication modulo \( 2^{16}+1 \) operation between key and plaintext sub-blocks.
6. Intra sub-block rotation on chunks of key sub-blocks.
7. Addition modulo \( 2^{16} \) operation between key and plaintext sub-blocks.
8. Inter sub-block rotation on key sub-blocks.
9. Multiplication modulo \( 2^{16}+1 \) operation between key and plaintext sub-blocks.
10. Intra sub-block rotation on chunks of key sub-blocks.
11. Addition modulo \( 2^8 \) operation between key and plaintext sub-blocks.
12. Exchange sub-block operation on key sub-blocks.
13. Multiplication modulo \( 2^{16}+1 \) operation between key and plaintext sub-blocks.
14. Intra sub-block rotation on key sub-blocks.
15. Modulo addition operation between key and plaintext sub-blocks.

For all normal implementation of PACMA, single round execution is sufficient as it provides the required security strength, but more rounds should be implemented for computing systems with higher throughput and processing capacities. Utilization of modulo arithmetic operations makes PACMA more computation intensive rather than communication intensive between the different processing elements available for execution in parallel environments.
5. Implementation and Analysis

PACMA is implemented in shared memory computing system with multi-core architecture using MPI, OpenMP and Java Thread programming with different sub-block sizes and compared with the sequential results. The speedup of various combinations of executions are analyzed and compared and the results are given in Table 1.

<table>
<thead>
<tr>
<th>SUB-BLOCK SIZE</th>
<th>SPEEDUP IN ECB MODE</th>
<th>Encryption</th>
<th>Decryption</th>
<th>Encryption</th>
<th>Decryption</th>
<th>Encryption</th>
<th>Decryption</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 bits</td>
<td>2.85</td>
<td>2.89</td>
<td>2.53</td>
<td>2.38</td>
<td>2.18</td>
<td>2.22</td>
<td></td>
</tr>
<tr>
<td>16 bits</td>
<td>3.42</td>
<td>3.47</td>
<td>2.57</td>
<td>2.63</td>
<td>2.41</td>
<td>2.46</td>
<td></td>
</tr>
<tr>
<td>32 bits</td>
<td>3.74</td>
<td>3.79</td>
<td>2.96</td>
<td>3.02</td>
<td>2.75</td>
<td>2.81</td>
<td></td>
</tr>
<tr>
<td>64 bits</td>
<td>4.31</td>
<td>4.38</td>
<td>3.37</td>
<td>3.43</td>
<td>3.17</td>
<td>3.22</td>
<td></td>
</tr>
<tr>
<td>128 bits</td>
<td>4.53</td>
<td>4.57</td>
<td>3.72</td>
<td>3.68</td>
<td>3.56</td>
<td>3.61</td>
<td></td>
</tr>
<tr>
<td>256 bits</td>
<td>4.82</td>
<td>4.87</td>
<td>3.91</td>
<td>3.95</td>
<td>3.74</td>
<td>3.79</td>
<td></td>
</tr>
</tbody>
</table>

ECB Mode: Electronic Code Book Mode
ENC : Encryption
DEC : Decryption

The performance of a cryptographic algorithm in parallel computing environment can be indicated using speedup. Speedup is the ratio of the time taken by the serial implementation of the algorithm to that of its parallel implementation and is denoted by \( S_p = \frac{T_s}{T_p} \). Where \( p \) denotes the parallel implementation with \( p \) number of processors or cores and \( s \) denotes its sequential implementation. The sequential implementation of PACMA is done using C programming language and is run on a single processor machine with single core.

When PACMA is implemented in a machine with multi-core processor with 4 cores, it yielded a better speedup for MPI implementation rather than OpenMP and Java thread implementations, as process implementations behaves better than threads for computation intensive operations in shared memory architectures. All the parallel implementations provided similar variation in their output. When the subblock size is kept small the speedup is low, but it gradually increased linearly when the sub-block size is increased. The decryption process provided better speedup than the encryption process because most of the values and the decisions computed for the encryption stages are made available to the decryption stages. A comparative representation of the performance of encryption algorithm using MPI, OpenMP and Java threads are shown in Fig. 4 and that of the decryption algorithm is shown in Fig 5.

The advantages of PACMA are its adaptive nature, its ability to run on different parallel computing architectures efficiently, its flexibility in deciding the size of the key and plain text sub-blocks and its ability to expand by suitably deciding the number of rounds. The level of Intra-packet parallelization, variety in grain size and the required security strength are also achieved by suitably deciding the sub-block size. Flow of the algorithm is made dynamic by determining the execution steps through each key value at the runtime.

6. Conclusion

PACMA is an adaptive cryptographic algorithm that provides better security strength and performance in parallel computing environments. It requires \( 5.7 \times 10^{288} \) years to break this cipher with brute force attack. PACMA is a dynamic algorithm as its granularity and execution stages are decided during runtime using the bit pattern in the key. As the general reversible techniques are used, this algorithm is scalable. The algorithm is exclusively designed for software implementations and hence it can be implemented over any parallel computing environment without much modification. The algorithm does not carry over any operation and hence it avoids dependency problem in the parallel processing environments.
References


