Islamic University of Gaza

Deanery of Higher Studies

Faculty of Engineering

Computer Engineering Department

Master Thesis

A HIGH PERFORMANCE ENHANCED SEQUENTIAL AND PARALLEL AES

تحسين أداء خوارزمية تشفير مطورة باستخدام البرمجة المتوازية

Fadi El-Faleet

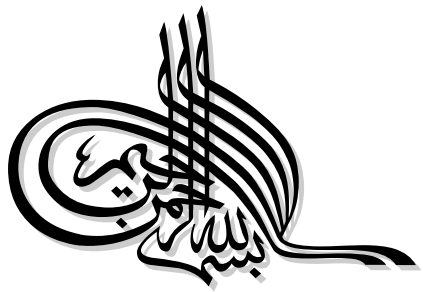
Supervisor

Prof. Mohammad A. Mikki

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science in Computer Engineering

Gaza, Palestine

(1432, 2011)



﴿ قَالُوا سُبْحَانَكَ لا عِلْمَ لَنَا إِلا مَا

عَلَّمْتَنَا إِنَّكَ أَنْتَ العَلِيْمُ الحَكِيْمُ﴾

[البقرة آية : 32]

Dedication

- ***To my beloved mother…***
- ***To my beloved father…***
- ***To my sisters and brothers…***
- ***To my wife…***
- ***To all friends…***

Acknowledgements

Praise is to Allah, the Almighty for having guided me at every stage of my life.

This thesis is the result of years of work whereby I have been accompanied and supported by many people. It is wonderful that I now have the opportunity to express my gratitude to all of them.

This work would not have been possible without the constant encouragement and support I received from prof. Mohammad Mikki, my advisor and mentor. I would like to express my deep and sincere gratitude to him. His understanding and personal guidance have provided a good basis for the present thesis.

I also extend my thanks to prof. Hatem Hamad and Dr. Aiman Abusamra the members of the thesis discussion committee.

Also, I would like to take this opportunity to express my profound gratitude to my beloved family – my mother, father, brothers, and my wife- without whom I would ever have been able to achieve so much. I especially wish to express my love for my mother, my father, my brothers and my wife, they did not only endure my manifold activities but also provided inspiration and support for my inclination to perfectionism. Only they know how much I am indebted to them.

Last, but certainly not least, I want to thank my friends, for their moral support during this study.

Table Of Contents

List Of Figures

List Of Tables

Contents **Page No.**

Chapter 4: The AES Construction

Abstract

In this thesis we discuss how to make a balance between the key size and the speed of the encryption algorithm such that the encryption algorithm has high speed and strong encryption. Since security always comes at a cost of performance.

We propose to improve the Advanced Encryption Standard with key size length of 128 bit. The improvement includes the following: combination between time and password, using secure hash, using message authentication code algorithms, executing the AES in parallel, determining the parallel task size as a percentage of the original data file, and finally increasing the overlap between parallel execution and communication (pipelining).

The contribution of the thesis includes enhancing the sequential AES and developing an original parallel AES based on the enhanced sequential AES.

Measurement results of the proposed parallel AES show that we have the optimal parallel task size between 15% and 25% of the data file. This task size gives us best performance (small parallel run time, small communication, and low power consumption).

Keywords: *Security, cryptography, Advanced Encryption Standard, Parallel Computing, Overlapping, speedup, Efficiency, Overhead Communication.*

تحسين أداء خوارزمية تشفير مطورة
باستخدام البرمجة المتوازية

ملخص البحث :

يقوم الباحث في هذه الأطروحة بمناقشة عملية التوازن بين طول المفتاح بالخوارزميات المشفرة بحيث تتمتع بسرعة عالية و قوة في عملية التشفير. عملية الأمن تأتي دائماً على حساب السرعة والأداء والعكس. لذلك نرغب بإيجاد توافق بين عامل السرعة والأداء وقوة التشفير.

أيضاً قدم الباحث مجموعة من التحسينات خاصة بتطوير خوارزمية محسنة للتشفير المتطور باستخدام مفتاح طول تشفيره 128 بت. هذا التحسين يشتمل على التالي: عملية دمج بين الوقت ومفتاح التشفير (كلمة السر)، استخدام خوارزمية التكرير الآمن، استخدام خوارزمية رسالة التأكيد الآمن، تشغيل خوارزمية التشفير المتقدم من خلال البرمجة المتوازية، تحديد حجم المهمة المنوي تشفيرها كنسبة من حجم الملف الكلي ، و في النهاية زيادة عملية التخطي بين التشغيل المتوازي والاتصالات.

وتكمن المساهمة الفعلية لهذه الدراسة في تطوير وتحسين خوارزمية التشفير المتقدم في حال البرمجة العادية ، واستخدام هذا التطوير بعد إدخال مجموعة من التحسينات في البرمجة المتوازية .

لقد بينت نتائج البحث بشكل كبير أن الحجم الأمثل للمهمة المنوي تشفيرها تكون محصورة بين 15% إلى 25% من حجم الملف الأصلي. هذا الحجم يعطينا أفضل أداء فيما يتعلق بتقليل الوقت المتوازي، وتقليل الاتصالات بين المعالجات بالإضافة إلى تقليل كمية الطاقة المستهلكة.

- كلمات مفتاحية: التشفير، التشفير المطور، عملية التخطي ، البرمجة المتوازية ، السرعة والفاعلية.

Chapter 1

Background Research

In this chapter, we introduce notation and basic principles in cryptography used in this thesis, the purpose of this chapter is to give a general idea of the principles, techniques, and algorithms which are required for understanding this thesis, but we rather assume the reader is familiar with cryptography.

1.1 Information Security and Cryptography

Security is one of the fundamental and important metrics in digital communication. Hence, cryptography which is one technique to ensure data is an important topic in computer security. Cryptography is a process of transmission of data through unsecured channels, and only the authenticated receiver who has the legitimate key can read the encrypted messages which might be documents, phone conversations, images or other forms of data as shown on Figure 1.1 .

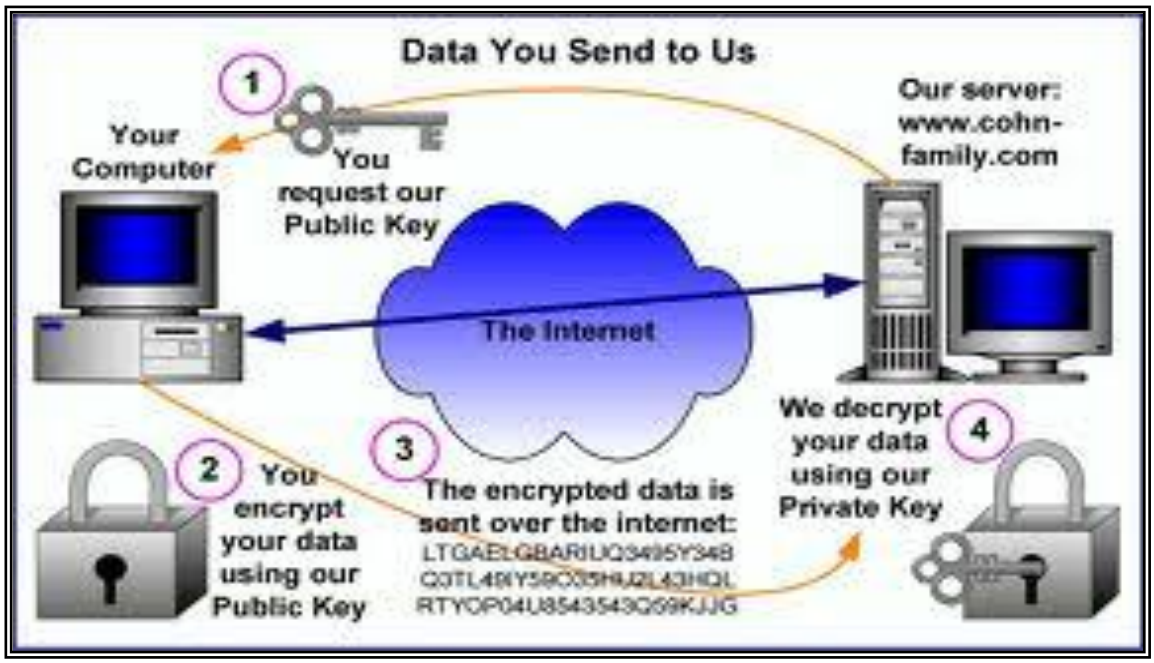


Figure 1.1: General Cryptosystem.

Definition 1.1: *Cryptography* is the study of mathematical techniques related to aspects of information security such as confidentiality, data integrity, entity authentication, and data origin authentication. Cryptography is not the only means of providing information security, but rather one set of techniques.

Definition 1.2: *Cryptography* is a general term referring to a set of cryptographic primitives used to provide information security services. Most often the term is used in conjunction with primitives providing confidentiality, i.e., encryption [1]. In cryptosystems, the information must be scrambled, so that other users will not be able to access the actual information. While providing privacy remains a central goal, the field has expanded to encompass many others, including not just other goals of communication security, such as guaranteeing integrity and authenticity of

communications, but many more sophisticated and fascinating goals. When you shop on the Internet, for example to buy a book, cryptography is used to ensure privacy of your credit card number as it travels from you to the shop's server. Also, in electronic banking, cryptography is used to ensure that your checks cannot be forged [2].

Cryptographic goals

Cryptography services must guarantee the following goals:

1. Confidentiality is a service used to keep the content of information from all but those authorized to have it. Secrecy is a term synonymous with confidentiality and privacy. There are numerous approaches to providing confidentiality, ranging from physical protection to mathematical algorithms which render data unintelligible.

2. Data integrity is a service which addresses the unauthorized alteration of data. To assure data integrity, one must have the ability to detect data manipulation by unauthorized parties. Data manipulation includes such things as insertion, deletion, and substitution.

3. Authentication is a service related to identification. This function applies to both entities and information itself. Two parties entering into a communication should identify each other. Information delivered over a channel should be authenticated as to origin, date of origin, data content, time sent, etc. for these reasons this aspect of cryptography is usually subdivided into major classes: entity authentication and data origin authentication. Data origin authentication implicitly provides data integrity. Data origin authentication implicitly provides data integrity (for if a message is modified, the source has changed).

4. Non-repudiation is a service which prevents an entity from denying previous commitments or actions. When disputes arise due to an entity denying certain actions were taken, a means to resolve the situation is necessary. For example. One entity may authorize the purchase of property by another entity and later deny such authorization is granted. A procedure involving a trusted third party is needed to resolve the dispute [1, 2].

The emergence of the Internet as a trusted medium for commerce and communication has made cryptography an essential component of modern information systems as shown in Figure 1.2. Cryptography provides the mechanisms necessary to implement accountability, accuracy, and confidentiality in communications [3]. As demands for secure communication bandwidth grow, efficient cryptographic processing will become increasingly vital to good system performance. To introduce cryptography, an understanding of issues related to information security in general is necessary. Information security manifests itself in many ways according to the situation and requirement. Regardless of who is involved, to one degree or another, all parties to a transaction must have confidence that certain objectives associated with information security have been met. Over the centuries, an elaborate set of protocols and mechanisms has been created to deal with information security issues when the information is conveyed by physical documents. Often the objectives of information security can not solely be achieved through mathematical algorithms and protocols alone, but require procedural techniques and abidance of laws to achieve the desired result. For example, privacy of letters is provided by sealed envelopes delivered by an accepted mail service. The physical security of the envelope is, for practical necessity,

limited and so laws are enacted which make it a criminal offense to open mail for which one is not authorized. It is sometimes the case that security is achieved not through the information itself but through the physical document recording it [4]. For example, paper currency requires special inks and materials to prevent counterfeiting. Conceptually, the way information is recorded has not changed dramatically over time. Whereas information was typically stored and transmitted on paper, much of it now resides on magnetic media and is transmitted via telecommunications systems. What has changed dramatically is the ability to copy and alter information. One can make thousands of identical copies of a piece of information stored electronically and each is indistinguishable from the original. With information on paper, this is much more difficult. What is needed then for a society where information is mostly stored and transmitted in electronic form is a means to ensure information security which is independent of the physical medium recording or conveying it and such that the objectives of information security rely solely on digital information itself [5].

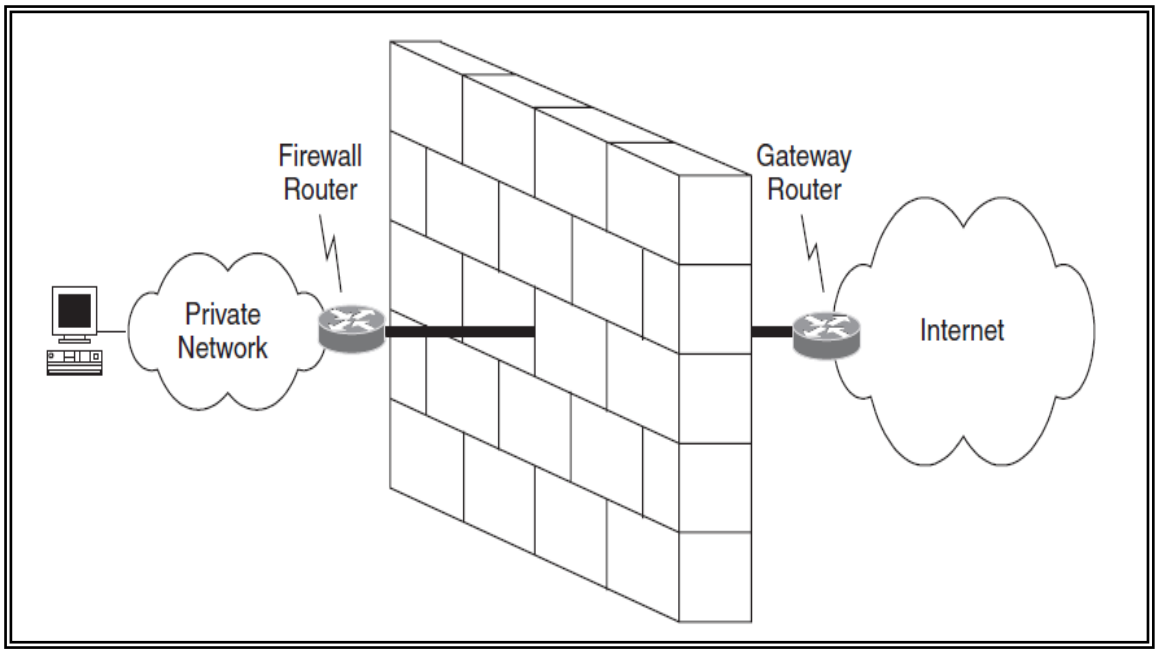


Figure 1.2: Network security system.

One of the fundamental tools used in information security is the signature. It is a building block for many other services such as non-repudiation, data origin authentication, identification, and witnessing, to mention a few. Having learned the basics in writing, an individual is taught how to produce a handwritten signature for the purpose of identification. At contract age the signature evolves to take on a very integral part of the person's identity. This signature is intended to be unique to the individual and serve as a means to identify, authorize, and validate. With electronic information the concept of a signature needs to be readdressed; it can not simply be something unique to the signer and independent of the information signed. Electronic replication of it is so simple that appending a signature to a document not signed by the originator of the signature is almost a triviality. Achieving information security in an electronic society requires a vast array of technical and legal skills. There is, however, no guarantee that all of the information security objectives deemed necessary can be adequately met. The technical means is provided through cryptography [6, 7].

1.2 Thesis Motivation

There is more and more information transmitted over networks every day. This information may be banks transactions, business information or military information which should be accessed by authorized persons. In order to protect the confidential information; secure cryptographic algorithms are required. There are two basic types of cryptosystems: Public key systems and private key systems. Public key systems are much slower than private key systems, but private key systems require key agreement through an existing secure channel [8].

Private key systems use the same key for both encryption and decryption operations. In order to communicate securely using a private key system, two parties must agree on the key using some pre-existing secure channel. When more than two parties are involved key distribution has been a major obstacle for practical uses of cryptography.

Public key cryptosystems help solve the key distribution problem by using separate keys for encryption and decryption operations, and making the encryption key public. Anyone can then encrypt a message, but only parties in possession of the private key can decrypt messages. Public key systems rely on one way trap door functions, which are interesting mathematical functions that can be easily computed in one direction but are very difficult to reverse unless a secret key is known [9].

The main goals of this thesis are

- *For the sequential AES-128 bit algorithm*
  1. We want to make a combination between time and password to generate the AES-128 bit key.
     a. The created AES key will be secure using SHA-256.
     b. We store the time and the message digest in the output file.
     c. So, to decrypt the file and make expansion of the key we must know the time to make the round keys.
     d. During data transfer on network we guarantee that if the data is changed the data file will not to be decrypted.

- *For the parallel AES-128 bit algorithm*
  1. We want to use multiprocessors to achieve the required speed to encrypt the large data file in parallel to minimize the total encryption time with lower power consumption.
  2. We want to determine the optimal parallel task size which is given to processors. This optimal parallel task size is a percentage of the original data file to achieve the following goals:
     - Load balance.
     - Small processor communication.
  3. Our proposed parallel AES algorithm is able to encrypt different extensions of files (video, text and images) with excellent quality since it deals with bits.

1.3 Research Questions

The importance of this study can be summarized by the following fundamental questions:

1. Why is it important to use the AES algorithm?
2. What is the impact of the combination between time and password to generate the round keys?
3. What is the best AES key size which gives good performance for parallel execution on multiprocessors with good security level?
4. What is the impact of using secure hash algorithm SHA-256?
5. What is the impact of determining the task size as a percentage of the original file?
6. What is the impact of parallel run time on power consumption?
7. How much does the proposed modified AES algorithm increase the security? Could it holdup against the different types of attacks?

1.4 Problem Definition

It is possible to implement cryptographic algorithms in software running on multiple processors. However, most of the cryptographic algorithms like DES (Data Encryption Standard) or 3DES have some drawbacks when implemented in software. for example, DES is no longer secure as computers get more powerful while 3DES is relatively sluggish in software. AES (Advanced Encryption Standard), which is rapidly being adopted worldwide, provides a better combination of performance and enhanced network security than DES or 3DES by being computationally more efficient than these earlier standards. Furthermore, by supporting large key sizes of 128, 192, and 256 bits, AES offers higher security against brute-force attacks [10]. Hence, we need fast and secure AES task granularity which requires little memory (so the algorithm can be used in smartcards) and give good quality at different types of file extensions.

The problem is divided into three steps as follows:

1. Find the AES task granularity that gives good performance on multi processors.
2. Increase the security of AES encryption by combining time and password and using secure hash algorithm to generate the keys.
3. Find the parallel task size as a percentage of the original file that gives good performance.

1.5 Thesis Research Scope

The research scope focuses on increasing the security level of AES, in addition to determining the parallel task size which gives low power consumption when working on multiprocessors.

1.6 Thesis Research Purpose

The objectives of the thesis are:

- Finding an efficient private key cryptosystem.
- Increasing the security of the private key by combining time and password to generate the encryption key and using the secure hash algorithm.
- Finding the parallel task size as a percentage of original data file size when executed on multiprocessors to get the following goals:
    - Load balance.
    - Small processor communication.
    - Good performance by decreasing the parallel run time that leads to low power consumption with excellent quality.

1.7 Research Methodology

The purpose here is to generate a robust, secure and fast cryptosystem, using different techniques, and different strategies, comparing the results with previous systems to determine the best. So, a variety of schemas will be tested to obtain the best combination between time and password and using secure hash algorithm to guarantee the data was not changed during transferring on network. We will determine the parallel task size as a percentage of original file to get high performance with low power consumption.

In other words, to answer these research goals, this research employs the research methodology in order to:

- Show the literature survey on cryptography.
- Propose efficient cryptosystem by:
    - Finding an efficient private key cryptosystem.
    - Increasing the security of the private key by combining time and password and using the secure hash algorithm.
    - Finding the parallel task size as a percentage of original data file size when executed on multiprocessors to get high performance.

1.8 Thesis Contributions

Practically everyone agrees that cryptography is an essential information security tool, and encryption can protect communications and stored information from unauthorized access and disclosure.

Cryptography allows people to carry over the confidence found in the physical world to the electronic world, thus allowing people to do business electronically without worries of deceit and deception. Everyday hundreds of thousands of people interact electronically, whether it is through e-mail, e-commerce, or ATM machines. The increase of information transmitted electronically has lead to an increased reliance on cryptography.

In this thesis, we develop an enhanced AES encryption algorithm with increased security by combining time and password and using the secure hash algorithm to guarantee that data is not changed during transfer over network. We also develop a parallel enhanced AES algorithm that paralyzes the developed enhanced AES. We run the parallel AES on multiprocessors using MPI to improve the performance (parallel time) and to decrease the power consumption.

The contributions of this thesis are as follows:

- We review several kinds of advanced encryption standard (AES).
- We select the AES with low work factor which gives good performance on multiprocessors.
- We increase the security of serial implementation of AES with low work factor which gives good security level.
- We determine the parallel task size as the percentage of the original data file size to get high performance with lower power consumption.
- We demonstrate through testing the efficiency and effectiveness of the proposed cryptosystem.

1.9 Outline of the Thesis

The rest of the thesis is organized as follows: Chapter 2 we discusses the principles and techniques of cryptography. Chapter 3 presents and evaluates numerous cryptography schemes and their performance, to avoid their weakness in our proposed cryptosystem. Chapter 4 discusses the construction of the advanced encryption system (AES) in detail. Chapter 5 presents the basic implementation of the parallel AES algorithm. It also describes speedup, parallel run time, efficiency, and communication overhead metrics to measure the performance of the parallel systems when execute parallel programs. Chapter 6 discusses our proposed hybrid encryption algorithm which increases security of AES by combining password and time to generate the key. Other proposed enhancements to AES include using secure hash algorithm, message authentication code. In addition, chapter 6 presents the proposed parallel AES algorithm which is based on the proposed enhanced sequential AES algorithm. High performance of the parallel AES comes from determining the optimal parallel task size as a percentage of the original data file size. Finally, chapter 7 concludes the thesis and presents suggested future work.

Chapter 2

Conventional Encryption

In this chapter we discuss the principles and techniques of cryptography**.**

Introduction

Cryptography is generally understood to be the study of the principles and techniques by which information is converted into an encrypted version that is difficult (ideally impossible) for any unauthorized person to convert to the original information, while still allowing the intended reader to do so. In fact, cryptography covers rather more than merely encryption and decryption. It is, in practice, a specialized branch of information theory with substantial additions from other branches of mathematics. Cryptography is probably the most important aspect of communications security and is becoming increasingly important as a basic building block for computer security [11].

There are, in general, two types of cryptographic schemes which are typically used to accomplish the cryptography goals. These two cryptographic schemes are secret key (or symmetric or conventional) cryptography and public-key (or asymmetric) cryptography. In symmetric-key cryptography, an algorithm is used to scramble the message using a secret key in such a way that it becomes unusable to all except the ones that have access to that secret key. The most widely known symmetric cryptographic algorithm is DES, developed by IBM in the seventies. It uses a key of 56 bits and operates on chunks of 64 bits at a time [10, 11].

In public key cryptography [12], algorithms use two different keys: a private key and a public one. A message encrypted with a private key can be decrypted with its public key (and vice versa). The owner of the key pair holds the private key, and may distribute the public key to anyone. Someone who wants to send a secret message uses the public key of the intended receiver to encrypt it. Only the receiver who holds the private key and can decrypt it.

The two basic building blocks of all encryption techniques are substitution and transposition. A substitution technique is one in which the letters of plaintext are replaced by other letters or by numbers or symbols. Transposition technique is a different kind of mapping where mapping is achieved by performing some sort of permutation on the plaintext letter [13].

2.1 Symmetric-key Cryptosystem

2.1.1 Principles of Symmetric-key Cryptosystem

In Symmetric-key cryptography, a single key is used for both encryption and decryption. As shown in Figure 2.1, the sender uses the key (or some set of rules) to encrypt the plaintext and sends the cipher text to the receiver. The receiver applies the same key (or rule set) to decrypt the message and recover the plaintext. Because a single key is used for both functions, secret key cryptography is also called symmetric encryption [14].

With this form of cryptography, it is obvious that the key must be known to both the sender and the receiver; that, in fact, is the secret. The biggest difficulty with this approach, of course, is the distribution of the key.

Symmetric-key cryptography schemes are generally categorized as being either stream ciphers or block ciphers [15]. Stream ciphers operate on a single bit (byte or computer word) at a time, and implement some form of feedback mechanism so that the key is constantly changing. A block cipher is so-called because the scheme encrypts one block of data at a time using the same key on each block. In general, the same plaintext block will always encrypt to the same cipher text when using the same key in a block cipher whereas the same plaintext will encrypt to different cipher text in a stream cipher.

Stream ciphers come in several flavors but two are widely used. Self synchronizing stream ciphers calculate each bit in the key stream as a function of the previous n bits in the key stream. It is termed "self-synchronizing" because the decryption process can stay synchronized with the encryption process merely by knowing how far into the n-bit key stream it is. One problem is error propagation; a garbled bit in transmission will result in n garbled bits at the receiving side. Synchronous stream ciphers generate the key stream in a fashion independent of the message stream but by using the same key stream generation function at sender and receiver. While stream ciphers do not propagate transmission errors, they are by their nature, periodic the key stream will eventually repeat [16].

2.2 Block ciphers

Block ciphers can operate in one of several modes; the following four are the most important [17]:

- **Electronic Codebook (ECB)** mode is the simplest, most obvious application: the secret key is used to encrypt the plaintext block to form a cipher text block. Two identical plaintext blocks, then, will always generate the same cipher text block. Although this is the most common mode of block ciphers, it is susceptible to a variety of brute-force attacks.

- **Cipher Block Chaining (CBC)** mode adds a feedback mechanism to the encryption scheme. In CBC, the plaintext is exclusively-ORed (XORed) with the previous cipher text block prior to encryption. In this mode, two identical blocks of plaintext never encrypt to the same cipher text.

- **Cipher Feedback (CFB)** mode is a block cipher implementation as a self synchronizing stream cipher. CFB mode allows data to be encrypted in units smaller than the block size, which might be useful in some applications such as encrypting interactive terminal input. In case of 1-byte CFB mode, for example, each incoming character is placed into a shift register the same size as the block, encrypted, and the block transmitted. At the receiving side, the cipher text is decrypted and the extra bits in the block (i.e., everything above and beyond the one byte) are discarded.

- **Output Feedback (OFB)** mode is a block cipher implementation conceptually similar to a synchronous stream cipher. OFB prevents the same plaintext block from generating the same cipher text block by using an internal feedback mechanism that is independent of both the plaintext and cipher text bit streams.

2.3 Model of Symmetric-key Cryptosystem

A symmetric or conventional encryption scheme *has five ingredients* [18] (Figure 2.3):

- **Plaintext /Message:** This is the original intelligible message or data that is fed into the algorithm as input.
- **Encryption Algorithm:** The encryption algorithm performs various substitution and transformation on the plaintext.
- **Secret Key:** The secret key is also the input to the encryption algorithm. The key is a value independent of the plaintext. The algorithm will produce a different output depending on the specific key being used at the time. The exact substitutions and transformations performed by the algorithm depend on the key.
- **Cipher text:** This is the scrambled message produced as output. It depends on the plaintext and secret key. For a given message, two different keys will produce two different cipher texts.
- **Decryption Algorithm:** This is essentially the encryption algorithm run in reverse. It takes the cipher text and the secret key and produces the original plaintext.

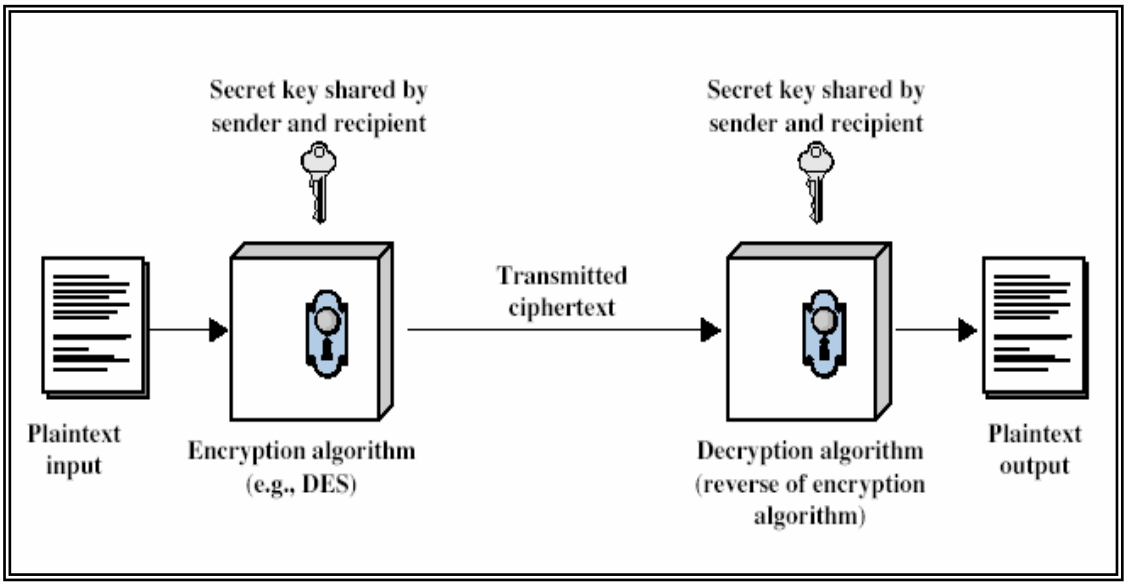


Figure 2.3: Model of Symmetric-key Cryptosystem.

There are two requirements for the secure use of symmetric-key encryption:

1. A strong encryption algorithm: At a minimum, the algorithm to be such that an opponent who knows the algorithm and has access to one or more cipher texts would be unable to decipher the cipher text or figure out the key. This requirement is usually stated in a stronger form: The opponent should be unable to decrypt cipher text or discover the key even if he or she is in possession of a number of cipher texts together with the plaintext that produced each cipher text [19].

2. Sender and receiver must have obtained copies of the secret key in a secure fashion and must keep the key secure. If some can discover the key and knows the algorithm, all communication using this key is readable.

In symmetric-key cryptosystem, a source produces a message in plaintext, **X =[X1, X2, X3, ..., XM]**. The M elements of X are letters in some finite alphabet.

Traditionally, the alphabet usually consists of the 26 capital letters. The binary alphabet **{0,1}** is also typically used. For encryption, a key of the form **K = [K1, K2, K3, ..., KJ]** is generated. If the key is generated at the message source, then it must also be provided to the destination by means of some secure channel.

Alternatively, a third party could generate the key and securely deliver it to both source and destination. With the message X and the encryption key K as input, the encryption algorithm forms the cipher text **Y =[Y1, Y2, Y3,. . . . ., YN].** This process can be expressed using the following notation:

$$Y = Ek(X)$$

This notation indicates that Y is produced by using the encryption algorithm E as a function of the plaintext X, with the specific function determined by the value of the key. The intended receiver, in possession of the key, is able to invert the transformation [20, 21]:

$$X = Ek(Y)$$

2.4 Importance of Symmetric-key Cryptography

The primary advantage of public-key cryptography is increased security and convenience. Private keys never need to transmitted or revealed to anyone. In a symmetric-key system, by contrast, the symmetric keys must be transmitted (either manually or through a communication channel), and there may be a chance that an enemy can discover the symmetric keys during their transmission [22].

Another major advantage of public-key systems is that they can provide a method for digital signatures. Authentication via symmetric-key systems requires the sharing of some symmetric keys and sometimes requires trust of a third party as well. As a result, a sender can repudiate a previously authenticated message by claiming that the shared symmetric key was somehow compromised by one of the parties sharing the symmetric-key. Public-key authentication, on the other hand, prevents this type of repudiation; each user has sole responsibility for protecting his or her private key. This property of public-key authentication is often called no repudiation [23].

A disadvantage of using public-key cryptography for encryption is speed; there are popular symmetric-key encryption methods that are significantly faster than any currently available public-key encryption method. Nevertheless, public-key cryptography can be used with symmetric-key cryptography to get the best of both worlds. For encryption, the best solution is to combine public - and symmetric-key systems in order to get both the security advantages of public-key systems and the speed advantages of symmetric-key systems. The public-key system can be used to encrypt a symmetric- key which is used to encrypt the bulk of a file or message. Such a protocol is called a digital envelope [23, 24].

In some situations, public-key cryptography is not necessary and symmetric-key cryptography alone is sufficient. This includes environments where secure symmetric key agreement can take place, for example by users meeting in private. It also includes

environments where a single authority knows and manages all the keys (e.g., a closed banking system). Since the authority knows everyone's keys already, there is not much advantage for some to be "public" and others "private." Also, public-key cryptography is usually not necessary in a single-user environment. In general, public-key cryptography is best suited for an open multi-user environment [25].

Public-key cryptography is not meant to replace symmetric-key cryptography, but rather to supplement it, to make it more secure. The first use of public-key techniques was for secure key exchange in an otherwise symmetric-key system; this is still one of its primary functions. Symmetric-key cryptography remains extremely important and is the subject of ongoing study and research [26].

Advantages of symmetric-key cryptography

1. Symmetric-key ciphers can be designed to have high rates of data throughput.
2. Keys for symmetric-key ciphers are relatively short.
3. Symmetric-key ciphers can be employed as primitives to construct various cryptographic mechanisms including pseudorandom number generators, hash functions, and computationally efficient digital signature schemes, to name just a few.
4. Symmetric-key ciphers can be composed to produce stronger ciphers. Simple transformations which are easy to analyze, but on their own weak, can be used to construct strong product ciphers.

Disadvantages of symmetric-key cryptography

1. In a two-party communication, the key must remain secret at both ends.
2. In a large network, there are many key pairs to be managed. Consequently, effective key management requires the use of an unconditionally trusted TTP.
3. In a two-party communication between n entities A and B, sound cryptographic practice dictates that the key be changed frequently and perhaps for each communication session. Digital signature mechanisms arising from symmetric-key encryption typically require either large keys for the public verification function or the use of a TTP.

Advantages of public-key cryptography

1. Only the private key must be kept secret (authenticity of public keys must, however, be guaranteed).
2. Depending on the mode of usage, a private key/public key pair may remain unchanged for considerable periods of time, e.g., many sessions (even several years).
3. Many public-key schemes yield relatively efficient digital signature mechanisms. The key used to describe the public verification function is typically much smaller than for the symmetric-key counterpart.
4. In a large network, the number of keys necessary may be considerably smaller than in the symmetric-key scenario.

Disadvantages of public-key encryption

1. Throughput rates for the most popular public-key encryption methods are several orders of magnitude slower than the best-known symmetric-key schemes.
2. Key sizes are typically much larger than those required for symmetric-key encryption, and the size of public-key signatures is larger than that of tags providing data origin authentication from symmetric-key techniques.

Summary of comparison

1. Public-key cryptography facilitates efficient signatures (particularly no repudiation) and key management.
2. Symmetric-key cryptography is efficient for encryption and some data integrity applications.

2.5 Cryptographic Hash Functions

Cryptographic Hash function is a complex encryption algorithm used in cryptography and it refers to a shortened version of full-scale encryption. A Cryptographic hash function is a deterministic procedure that takes an arbitrary block of data and returns a fixed-size bit string, the hash value, such that an accidental or intentional change to the data will change the hash value. The data to be encoded is often called the message. The hash value is sometimes called message digest or simply digest [27].

Ideal Characteristics of Cryptographic Hash Functions

1. It is easy to compute the hash value for any given message.
2. Cryptographic hash functions are infeasible to generate a message that has a given hash.
3. It is not possible to modify a message without changing the hash in cryptographic hash functions.
4. In cryptographic hash functions, two different messages cannot be with same hash.

Types of Cryptographic Hash Functions are: MAC, MDC, CRHF (collision resistant hash functions), UOWHF (universal one-way hash functions), and OWHF (one-way hash functions). Some of the popular Cryptographic Hash Functions are: SHA- Secure Hash Algorithms (SHA-0, SHA-1), MD- Message Digest Algorithms (MD5, which has 128-bit hash value), and RIPEMD- Race Integrity Primitives Evaluation Message Digest Algorithms (RIPEMD-128 and RIPEMD-160). Of all these, the most commonly used cryptographic hash functions are MD5 and SHA-1.

Applications of Cryptographic Hash Functions:

1. Cryptographic hash functions are used in the verification of message integrity.
2. These hash functions allows a fast look-up of the data in a hash table.

3. Cryptographic hash functions use peer-to-peer file sharing networks to identify files.
4. These hash functions are used in contexts where it is necessary for the users to protect themselves against the possibility of forgery.
5. Cryptographic hash functions are also used in the generation of pseudorandom bits, to derive new keys or passwords from a single, secure key or password.
6. These functions are widely used in information authentication.
7. These cryptographic hash functions are used to maintain secrecy of the client password and hashes must be kept valid for a session.
8. Cryptographic hashes provide security for E-mail and file transfer systems.
9. Cryptographic Hash Functions are also used in Database matching and software downloads.

2.6 Summary

This chapter covers the fundamentals and terminologies of cryptography, including the issues of private key cryptosystems and public key cryptosystems. We mention the advantages and disadvantages of symmetric-key cryptosystems, which motivate us to propose our enhanced system.

Chapter 3

Literature Review

In this chapter numerous cryptography schemes are studied and their performance is evaluated, to avoid their weakness in our proposed cryptosystem.

3.1 Previous Work

Most cryptographic algorithms function more efficiently when implemented in hardware than in software running on single processor. However, systems that use hardware implementations have significant drawbacks: they are unable to respond to flaws discovered in the implemented algorithm or to changes in standards. As an alternative, it is possible to implement cryptographic algorithms in software running on multiple processors. However, most of the cryptographic algorithms like DES (Data Encryption Standard) or 3DES have some drawbacks when implemented in software: DES is no longer secure as computers get more powerful while 3DES is relatively sluggish in software. AES (Advanced Encryption Standard), which is rapidly being adopted worldwide, provides a better combination of performance and enhanced network security than DES or 3DES by being computationally more efficient than these earlier standards. Furthermore, by supporting large key sizes of 128, 192, and 256 bits, AES offers higher security against brute-force attacks [28].

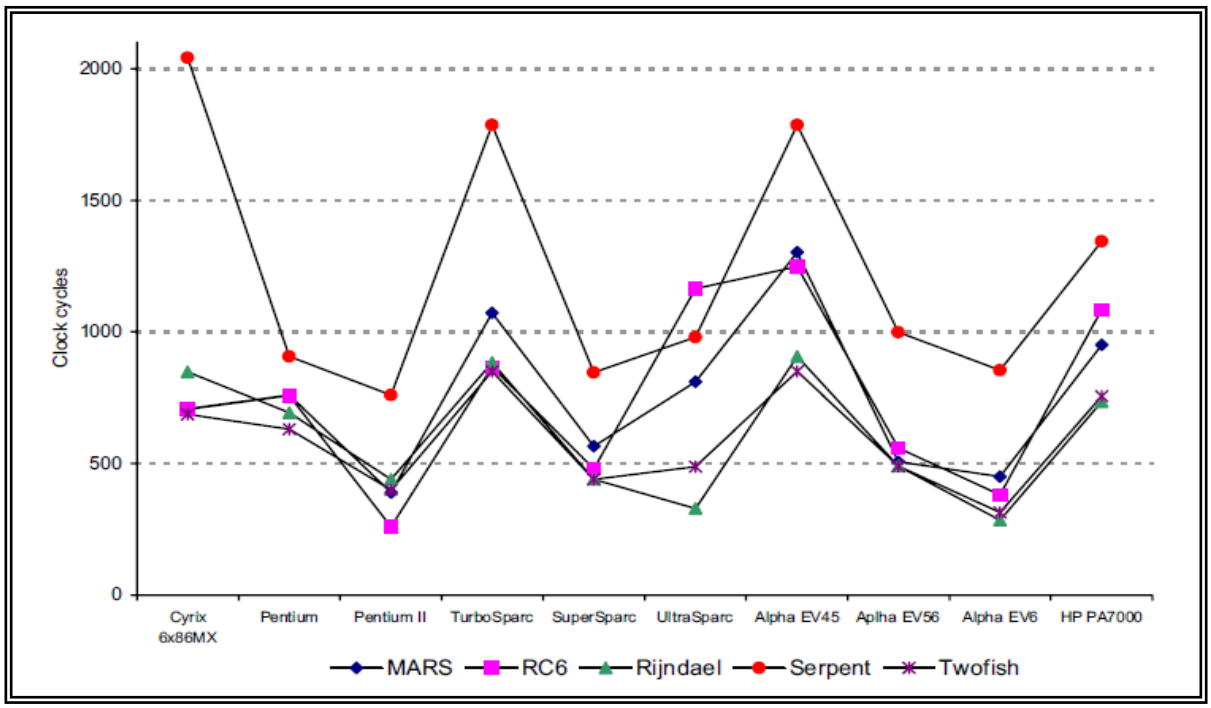


Figure 3.1: Encryption speeds for 128 bits on different processors.

In [30] the author illustrates the different work factors of AES on different multiprocessors and finds that the work factor of 128 bit gives us best performance as shown in Figure 3.2.

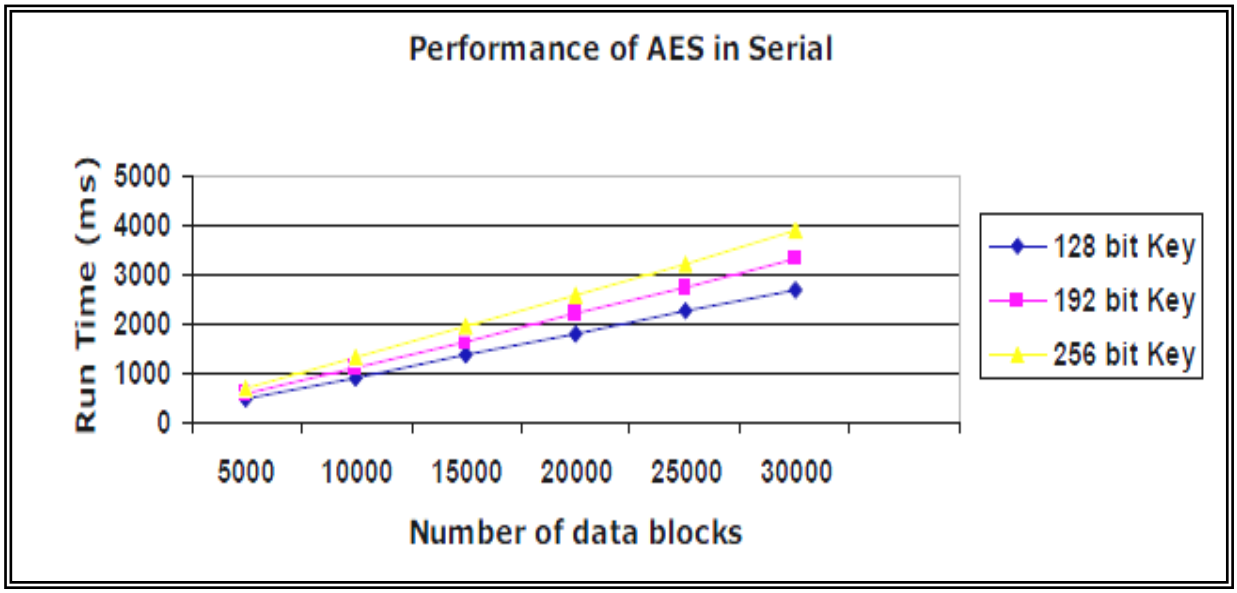


Figure 3.2: Encryption speeds for 128 bits.

In [31], Biham introduces the notion of comparing the algorithms based on their ”minimal secure variants.” Different design teams were more or less conservative than each other; the number of rounds they could successfully cryptanalze. Biham tries to normalize the algorithms by determining the minimal number of rounds that is secure and then adds a standard two cycles. The results are shown in Figure 3.3.

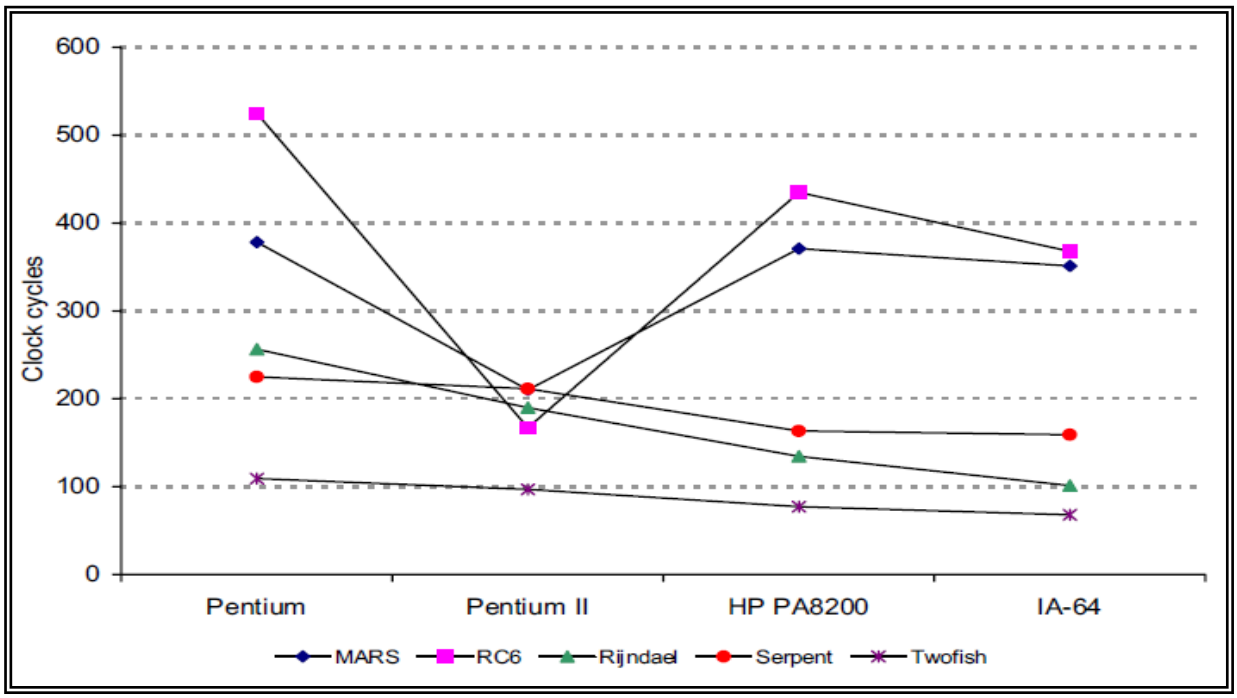


Figure 3.3: Encryption speeds for the Minimal secure Variant In Assembly.

Encryption with AES is based on a secret key with 128, 192 or 256 bits. But if the key is easy to guess it doesn’t matter if AES is secure, so it is as critically vital to use good and strong keys as it is to apply AES properly. Creating good and strong keys is a surprisingly difficult problem and requires careful design when done with a computer. Keys derived into a fixed length suitable for the encryption algorithm from

passwords or pass phrases typed by a human will seldom correspond to 128 bits much less 256. To even approach 128-bit equivalence in a pass phrase, at least 10 typical passwords of the kind frequently used in day-to-day work are needed. Weak keys can be somewhat strengthened by special techniques by adding computationally intensive steps which increase the amount of computation necessary to break it. The risks of incorrect usage, implementation and weak keys are in no way unique for AES; these are shared by all encryption algorithms. [32].

For some of the candidates, the performance or RAM requirements can depend on whether encryption or decryption is being performed. Many smart card terminals contain a secure module. In several applications, it is a requirement that two smart cards execute a protocol together, and many existing protocols use both encryption and decryption on the same smart card [33].

In [62] the author illustrates the difference between the AES and other algorithms and discusses a number of criteria:

- How **secure** the algorithm is currently judged to be in the cryptographic literature;
- The **performance** characteristics of the algorithm (e.g. the "raw speed" of the algorithm, and whether it supports parallel encryption);
- How **politically safe** a decision it is to use a particular algorithm (paradoxically, this doesn't necessarily depend directly on the algorithm's security);
- Whether you have to interact with a **legacy system**.

The authors compare measured speed of encryption with various algorithms available as standard in Sun's JDK. The encryption algorithms authors consider here are AES (with 128 and 256-bit keys), DES, Triple DES, RC4 (with a 256-bit key) and Blowfish (with a 256-bit key). Figure 3.4 shows the time taken to encrypt various numbers of 16-byte blocks of data using the algorithms mentioned.

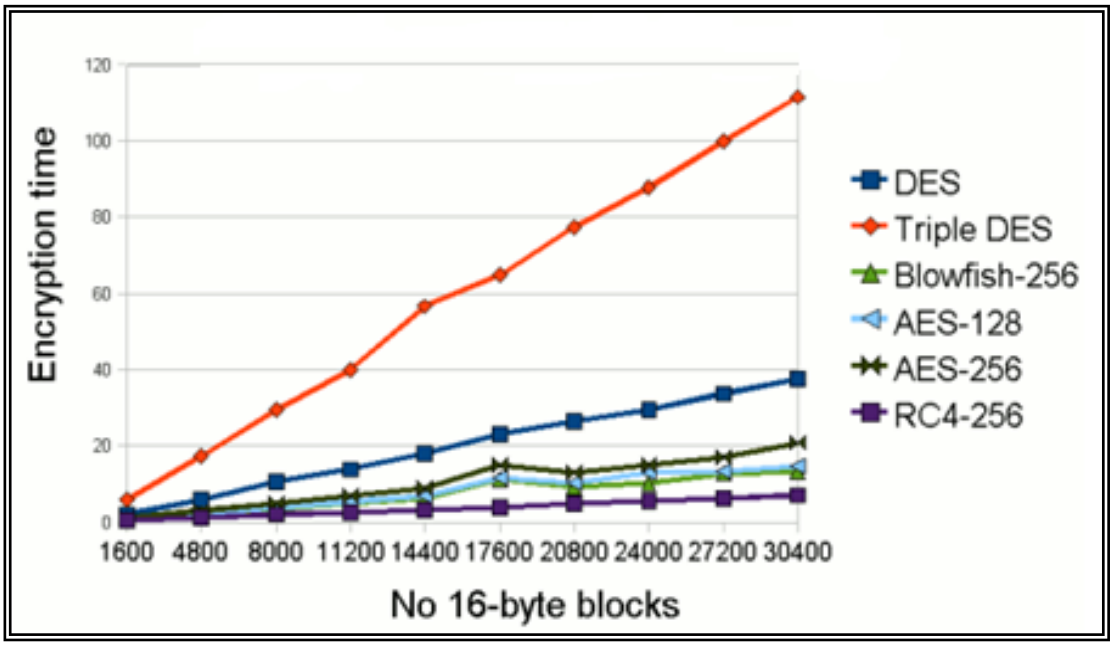


Figure 3.4: Cipher encryption speed.

Nobody yet has (publicly) a full attack on AES, or a partial attack that is practical (though some impractical partial attacks exist). However, AES is algebraically simpler than other block ciphers: effectively, it can be written as a series of mathematical equations, and there is a worry that somebody could demonstrate a way to solve those equations.

In [63] the authors illustrate the minimum required key size is some combination of the maximum key size that can be attacked now for a given algorithm, extrapolated to the number of years that you need to keep the encrypted data confidential. Needless to say, such extrapolation is extremely difficult beyond a few years.

In [63] the authors take data points from attacks on keys of various lengths and extrapolate via Moore's Law. They conclude that in 2030, it will seem as difficult to brute-force a 93-bit symmetric key as it did to brute-force DES in 1982, and that in 2050 this will be true for a 109-bit key. From this, we might conclude that, ignoring collision attacks, a 128-bit key is sufficient to keep data confidential for the next few decades. Taking into account collision attacks, in [64] the authors recommend using a 256-bits key size. In [65], presumably based on similar types of calculation, recommend a minimum of "128 bits of strength" to keep data confidential "beyond 2030".

The Visa credit card company's Data Field Encryption best practices guide suggest that for the purpose of transmitting credit card numbers, "keys should have the strength of at least 112 equivalent bit strength". For this purpose, they actually judge 128 bits (as in the minimum AES key size) to be "stronger than needed" (presumably because of the relatively short lifespan of credit cards).

In principle, a quantum computer could square root the effort of a brute-force key search, so that the bit strength of the key is halved (i.e. a 256-bit key in quantum computing has the strength of a 128-bit key in classical computing).

In [66] the author looks at using implementing a modified version of the AES algorithm to be optimized in the use of smart cards. Although not relevant necessarily to the research being undertaken, the authors went into great detail explaining how the AES algorithm works, which is of relevance to the research.

In [67] the author looks at a high speed/low cost implementation of the AES cipher, exploring different ways of physical wiring the AES operations to enhance performance of AES.

We want to deal with the weakness of AES and to increase its performance. *The main questions are as follows:*

1. Determining the AES work factor which requires little memory (so the algorithm can be used in smartcards efficiently) .
2. How can we increase the security level of AES without using more work factors?
3. How can we decrease the power consumption of AES to give good performance in smart card and other systems?
4. How can we increase the performance of AES especially in large files?

5. How to find agreement between the opposite criteria of AES security and performance.

3.2 Research Issues

We notice that AES algorithm works on different work sizes. AES with high work factor enjoys top security but suffers from low speed and consumes high power.

It's observed that AES encryption algorithm has one or more of *the following weaknesses:*

- Needs a key distribution method.
- Low encryption speed.
- High power consumption.
- Vulnerable to brute force attack.

So, we need a system that has the advantage of being secure and fast. Moreover, the proposed cryptosystem must overcome most of the previous shortcomings.

Chapter 4

The AES Construction

In this chapter we discuss the construction of Advanced Encryption System (AES) in details.

Introduction

The Rijndael proposal for AES [34] defined a cipher in which the block length and the key length specified to be 128, 192, or 256 bits. The AES specification uses the same three key size alternatives but limits the block length to 128 bits. A number of AES parameters (Table 4.1) depend on the key length. Most of the implementation of AES uses the key length of 128 bits.

Table (4.1): The AES Parameter.

Key size (words/byte/bits)	4/16/128	6/24/192	8/32/256
Plaintext block size (word/bytes/bits)	4/16/128	4/16/128	4/16/128
Number of rounds	10	12	14
Round key size (words/bytes/bits)	4/16/128	4/16/128	4/16/128
Expanded key size (words/bytes)	44/176	52/208	60/240

Requirements of AES

It had to satisfy certain engineering criteria:

- Performance, efficiency, implementability, and flexibility.
- Rijndael can be implemented easily in both hardware and software.
- Has realizations that require little memory (so the algorithm can be used in smartcards).

The comparison between the AES and other cryptography algorithms is given in table 4.2 [35]. By using metric points from (1 to 3) we will measure the strength of an cryptographic algorithm. So, Point 1 is low level, point 2 is medium level and point 3 is high level. The algorithm which sums more points is the best one as shown in table 4.2 [35].

Table (4.2): The Comparison between different Cryptography algorithms.

Algorithm Name	AES	Serpent	Twofish	MARS	RC6
General Security	2	3	3	3	2
Implementation Difficulty	3	3	2	1	1
Software Performance	3	1	1	2	2
Smart-Card Performance	3	3	2	1	1
Hardware Performance	3	3	2	1	2
Design Features	2	1	3	2	1
Total	**16**	**14**	**13**	**10**	**9**

Features of AES Encryption Algorithm:

- Advanced Encryption Standard (AES) algorithm works on the principle of Substitution Permutation network.
- AES is fast in both software and hardware.
- AES operates on a 4×4 matrix of bytes termed as a state
- The Advanced Encryption Standard cipher is specified as a number of repetitions of transformation sounds that convert the input plaintext into the final output of cipher text.
- Each round consists of several processing steps, including one that depends on the Encryption key.
- A set of reverse rounds are applied to transform cipher text back into the original plaintext using the same encryption key.

Advantages of AES Encryption Algorithm:

- Advanced Encryption Standard not only assures security but also improves the performance in a variety of settings such as smartcards, hardware implementations etc.
- AES is federal information processing standard and there are currently no known non-brute-force direct attacks against AES.
- AES is strong enough to be certified for use by the US government for top secret information.

Alternative to Advanced Encryption Standard: The ciphers which are used alternatively to Advanced Encryption Standard are SSl and TLS. RC4 encryption is next to AES. RC4 is of 128- bits RC4 is a fast cipher and is always subjected to many types

of attacks. That is the reason WEP wireless encryption is poor. Thus AES is given priority than other standards [35].

4.1 Description of the Cipher

The Advanced Encryption Standard is a block cipher that consists of four basic steps that are used in a series of 10 rounds to fully encrypt a 128-bit (16-byte) block of input data. The algorithm can encrypt any file or data size and will produce an encrypted output of identical size using an input cipher key (which can come from any of a number of public and/or private key methods). The AES algorithm successively encrypts the input data block in a square matrix representation to produce the output. There are five major components to the algorithm: (1) Key expansion; (2) the Add Round Key step; (3) the Sub Bytes step; (4) the Shift Rows step; and (5) the Mix Columns step [36].

- *Key Expansion (AES KEY 128 bit):*

Since the algorithm consists of 10 rounds of execution the Key Expansion step is used to create a unique encryption key for each of the 10 rounds [37].

```
Cipher Key:       00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Expansion[0]:     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Expansion[1]:     62 63 63 63 62 63 63 63 62 63 63 63 62 63 63 63
Expansion[2]:     9b 98 98 c9 f9 fb fb aa 9b 98 98 c9 f9 fb fb aa
Expansion[3]:     90 97 34 50 69 6c cf fa f2 f4 57 33 0b 0f ac 99
Expansion[4]:     ee 06 da 7b 87 6a 15 81 75 9e 42 b2 7e 91 ee 2b
Expansion[5]:     7f 2e 2b 88 f8 44 3e 09 8d da 7c bb f3 4b 92 90
Expansion[6]:     ec 61 4b 85 14 25 75 8c 99 ff 09 37 6a b4 9b a7
Expansion[7]:     21 75 17 87 35 50 62 0b ac af 6b 3c c6 1b f0 9b
Expansion[8]:     0e f9 03 33 3b a9 61 38 97 06 0a 04 51 1d fa 9f
Expansion[9]:     b1 d4 d8 e2 8a 7d b9 da 1d 7b b3 de 4c 66 49 41
Expansion[10]:    b4 ef 5b cb 3e 92 e2 11 23 e9 51 cf 6f 8f 18 8e
```

Figure 4.1: Key Expansion.

4.2 Add Round Key Step

One of the simplest steps in the AES algorithm, the Add Round Key step simply adds the 128-bit key for the particular round to the 128-bit block of data currently being used in the algorithm [38]. The 128-bit data is arranged as a 4x4 array of bytes (16 bytes in total).

4.3 Sub Bytes Step

In the Sub Bytes step, each byte in the data array is updated using an 8-bit S-Box to provide a certain amount of non-linearity to the AES cipher. The S-Box was designed as part of the original Rijndael algorithm to be resistant to linear and differential cryptanalysis for even great security within the AES algorithm [39]. There is a single S-Box for the AES algorithm and can be implemented in the form of a table lookup as shown in Figure. 4.3**.**

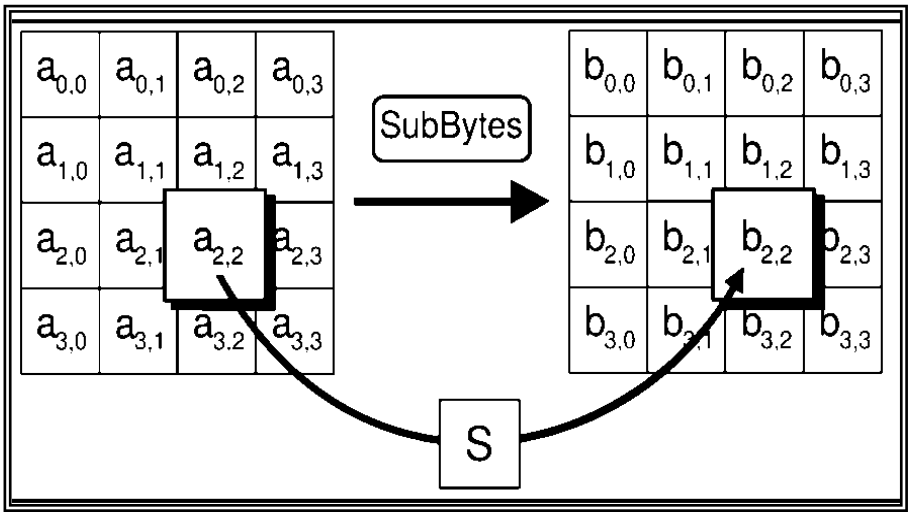


Figure 4.2: The Sub Bytes substitution step.

4.4 Shift Rows Step

The rows of the data matrix are cyclically shifted by a different offset for each row. The first row is left the same, while the second row is shifted by one byte to the left. The third and fourth rows are shifted by two and three bytes to the left, respectively [40]. In this way, the output in this step contains columns which consist of a mix of each of the input columns, further obscuring the complexity of the cipher as shown in Figure. 4.4.

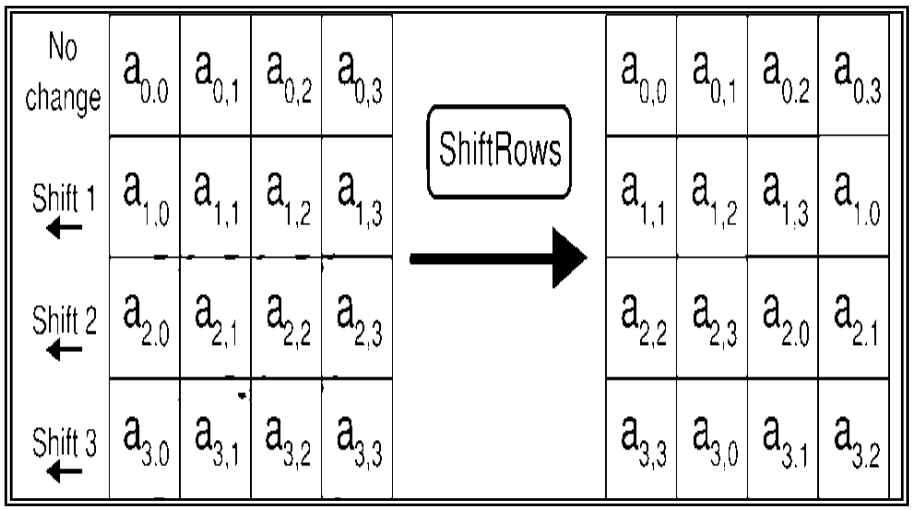


Figure: 4.3: The result of the Shift Rows step on the input data block.

4.5 Mix Columns Step

The four bytes of each column in the input array are combined using an invertible linear transformation where each byte of the input column has a direct effect on all four bytes in the output column. The primary purpose of this step is to further diffuse and complicate the output of the AES algorithm [41]. The linear transformation can be represented as multiplication by a fixed polynomial C(x) as shown in Figure. 4.5.

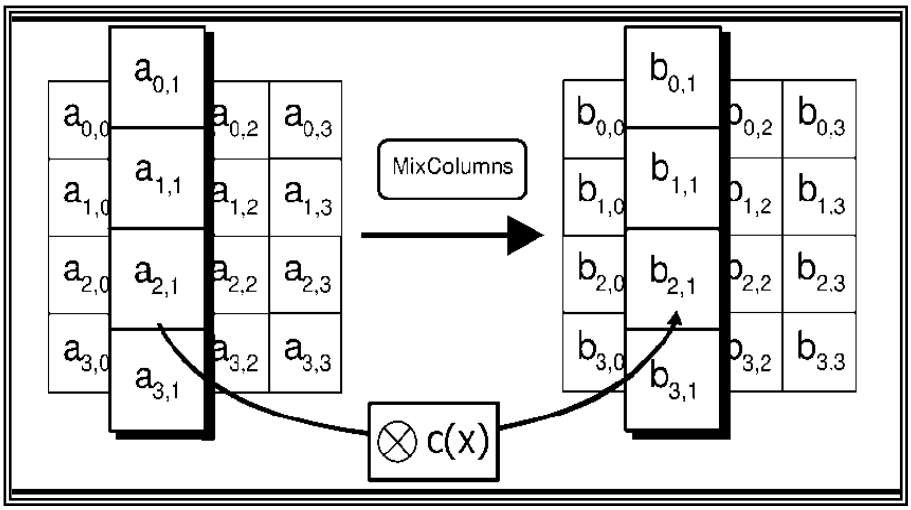


Figure 4.4: The result of the Mix Columns step.

4.6 High Level Description of The Algorithm

1. Key Expansion round keys are derived from the cipher key using Rijndael's key schedule .
2. *Initial Round:*
    - Add Round Key each byte of the state is combined with the round key using bitwise xor.
3. *Rounds:*
    - **Sub Bytes:** a non-linear substitution step where each byte is replaced with another according to a lookup table.
    - **Shift Rows:** a transposition step where each row of the state is shifted cyclically a certain number of steps.
    - **Mix Columns:** a mixing operation which operates on the columns of the state, combining the four bytes in each column.
    - Add Round Key.
4. *Final Round (no Mix Columns) :*
    1. Sub Bytes.
    2. Shift Rows.
    3. Add Round Key.

The encryption and decryption operations are shown in Figure. 4.5.

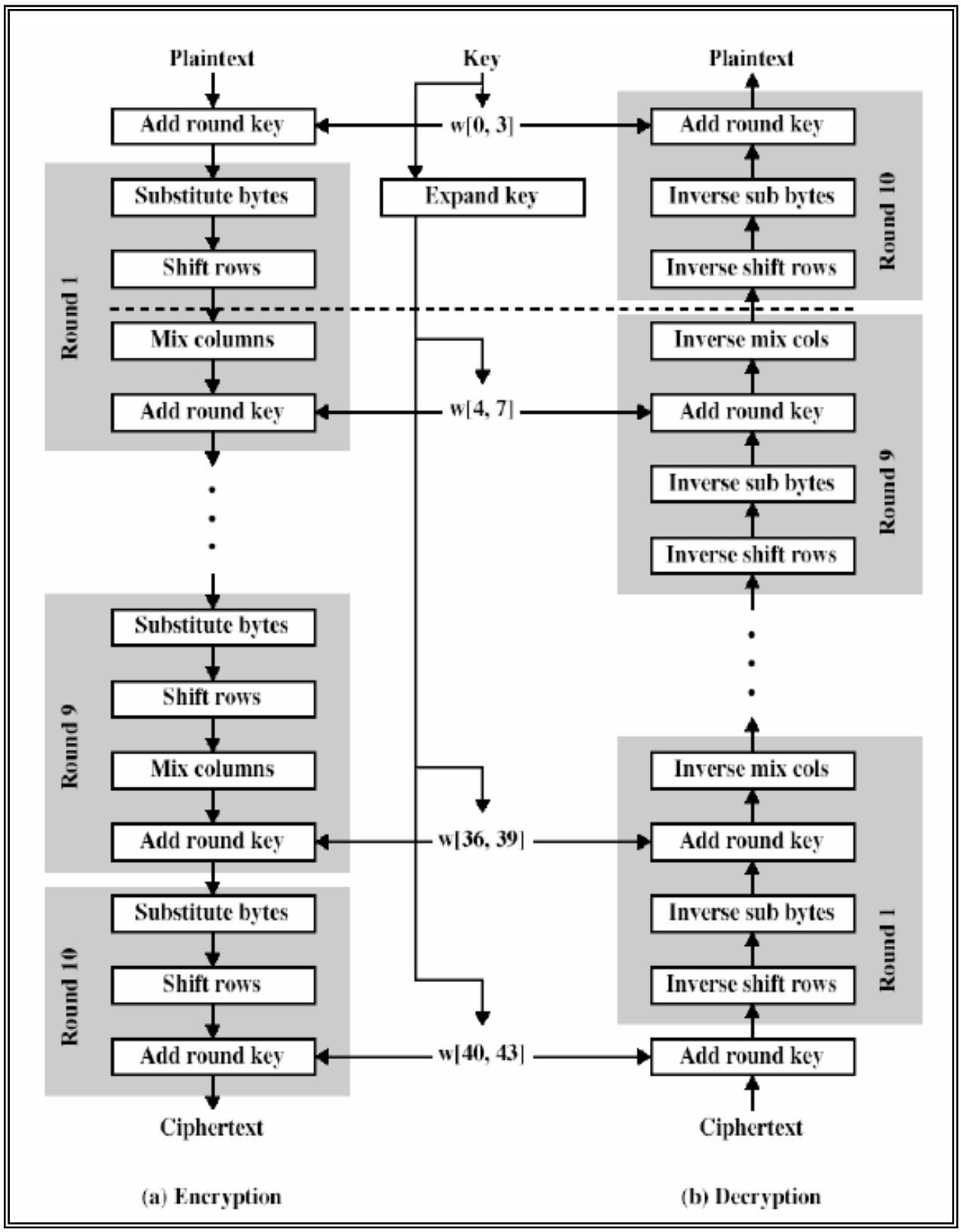


Figure 4.5: AES encryption and decryption.

4.7 Security

Until May 2009, the only successful published attacks against the full AES were side-channel attacks on some specific implementations. The National Security Agency (NSA) reviewed all the AES finalists, including Rijndael, and stated that all of them were secure enough for U.S. Government non-classified data. In June 2003, the U.S. Government announced that AES may be used to protect classified information [42].

The design and strength of all key lengths of the AES algorithm (i.e., 128, 192 and 256) are sufficient to protect classified information up to the SECRET level. TOP SECRET information will require use of either the 192 or 256 key lengths. The implementation of AES in products intended to protect national security systems and/or information must be reviewed and certified by NSA prior to their acquisition and use [43].

AES has 10 rounds for 128-bit keys, 12 rounds for 192-bit keys, and 14 rounds for 256-bit keys. By 2006, the best known attacks were on 7 rounds for 128-bit keys, 8 rounds for 192-bit keys, and 9 rounds for 256-bit keys.

4.8 Performance

High speed and low RAM requirements were criteria of the AES selection process. Thus AES performs well on a wide variety of hardware, from 8-bit smartcards to high-performance computers. On a Pentium Pro, AES encryption requires 18 clock cycles / byte, equivalent to a throughput of about 11 MB/s for a 200 MHz processor. On a Pentium M 1.7 GHz throughput is about 60 MB/s [44].

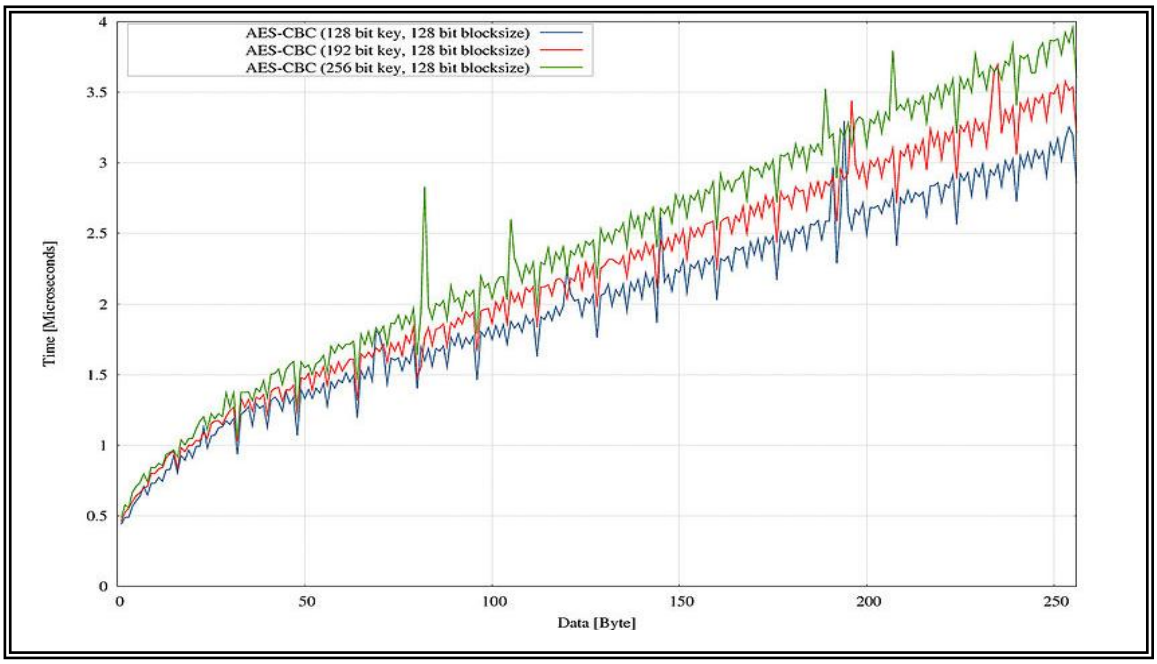


Figure 4.6: AES Block Cipher Speed.

4.9 Summary

Advanced Encryption Standard not only assures security but also improves the performance in a variety of settings such as smartcards, hardware implementations etc. AES is federal information processing standard and there are currently no known non-brute-force direct attacks against AES. AES is strong enough to be certified for use by the US government for top secret information.

Chapter 5

Parallel Implementation of AES

In this chapter we present the basic implementation of parallel AES algorithm. We also describe the speedup, run time, efficiency, and communication overhead metrics to measure the performance of parallel systems.

5.1 The Importance of parallel Computing Systems

The current trend in high performance computing is clustering and distributed computing. In clusters, powerful low cost workstations and/or PCs are linked through fast communication interfaces to achieve high performance parallel computing. Recent increases in communication speeds, microprocessor clock speeds, availability of high performance public domain software including operating system, compiler tools and message passing libraries, make cluster based computing appealing in terms of both high performance computing and cost effectiveness [45].

Parallel computing on clustered systems is a viable and attractive proposition due to the high communication speeds of modern networks. To efficiently use more than one processor in a program, the processors must share data and coordinate access to and updating of the shared data. The most popular approach to this problem is to exchange of data through messages between computers. The MPI (Message Passing Interface) approach is considered to be one of the most mature methods currently used in parallel programming mainly due to the relative simplicity of using the method by writing a set of library functions or an API (Application Program Interface) callable from C, C++ or Fortran Programs. MPI was designed for high performance on both massively parallel machines and clusters. For implementing the AES algorithm in parallel, the MPI based cluster is used in the this chapter. The performance of a parallel algorithms depend not only on input size but also on the architecture of the parallel computer, the number of processors, and the interconnection network. In this chapter, different types of parallel architectures are discussed before actually implementing the parallel algorithm of AES [46].

5.2 SIMD Architectures

SIMD (Single-Instruction Stream Multiple-Data Stream) [47] architectures are essential in the parallel world of computers. In SIMD architectures, several processing elements are supervised by one control unit. All the processing units receive the same instruction from the control unit but operate on different data sets, which come from different data flows, meaning that they execute programs in a lockstep mode, in which each processing element has its own data stream. There are two types of SIMD architectures: the True SIMD and the Pipelined SIMD. Each has its own advantages and disadvantages but their common attribute is superior ability to manipulate **vectors.** Figure 5.1 shows a model of an SIMD architecture.

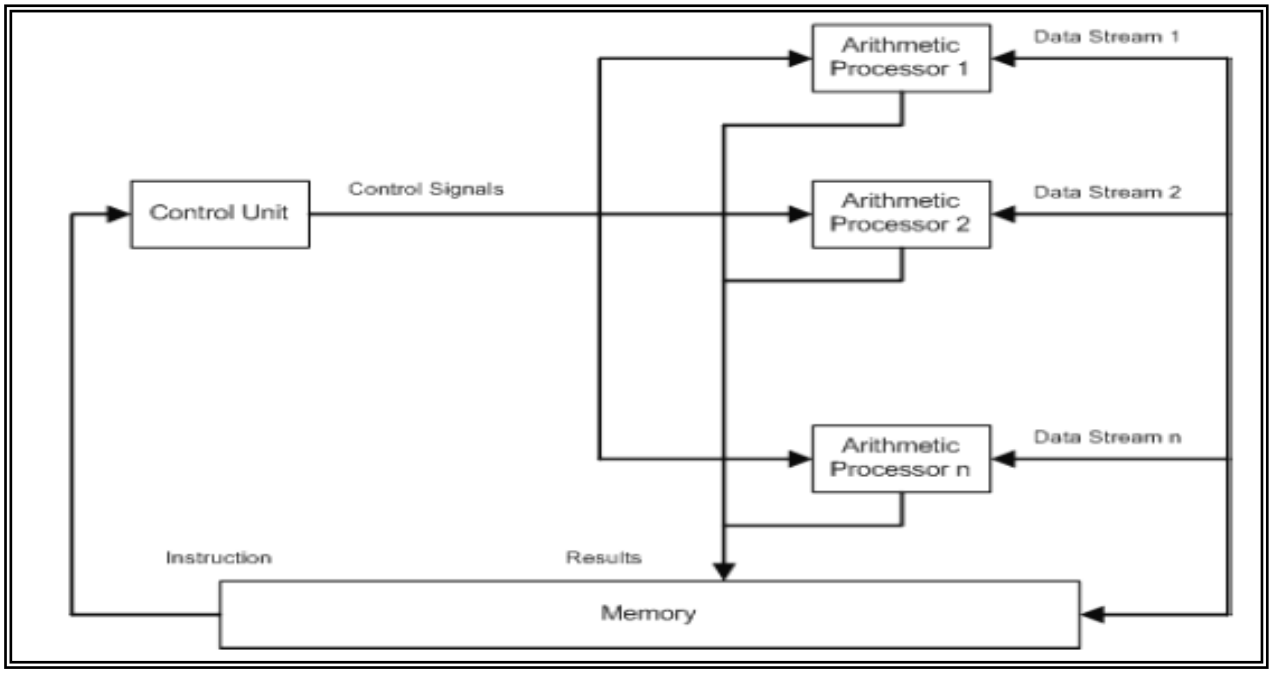


Figure 5.1: Model of an SIMD architecture.

The true SIMD architecture contains a single control unit (CU) with multiple processor elements (PE) acting as arithmetic units (AU). In this situation, the arithmetic units are slaves to the control unit. The AU's cannot fetch or interpret any instructions. They are merely a unit which has capabilities of addition, subtraction, multiplication, and division. Each AU has access only to its own memory. In this sense, if an AU needs the information contained in a different AU, it must put in a request to the CU and the CU must manage the transferring of information. The advantage of this type of architecture is in the ease of adding more memory and AU's to the computer. The disadvantage can be found in the time wasted by the CU managing all memory exchanges [48].

Pipelined SIMD architecture is composed of a pipeline of arithmetic units with shared memory. The pipeline takes different streams of instructions and performs all the operations of an arithmetic unit. The pipeline is a first in first out type of procedure. To take advantage of the pipeline, the data to be evaluated must be stored in different memory modules so the pipeline can be fed with this information as fast as possible. The advantages to this architecture can be found in the speed and efficiency of data processing [49].

5.3 MIMD Architectures

Multiple instruction stream, multiple data stream (MIMD) [50] machines have a number of processors that function asynchronously and independently. At any time, different processors may be executing different instructions on different pieces of data. MIMD architectures may be used in a number of application areas such as computer-aided design/computer-aided manufacturing, simulation, modeling, and as communication switches. MIMD machines can be of either shared memory or distributed memory categories. These classifications are based on how MIMD processors access memory. Shared memory machines may be of the bus-based,

extended, or hierarchical type. Distributed memory machines may have hypercube or mesh interconnection schemes. Figure 5.2 shows a Model of a MIMD architecture.

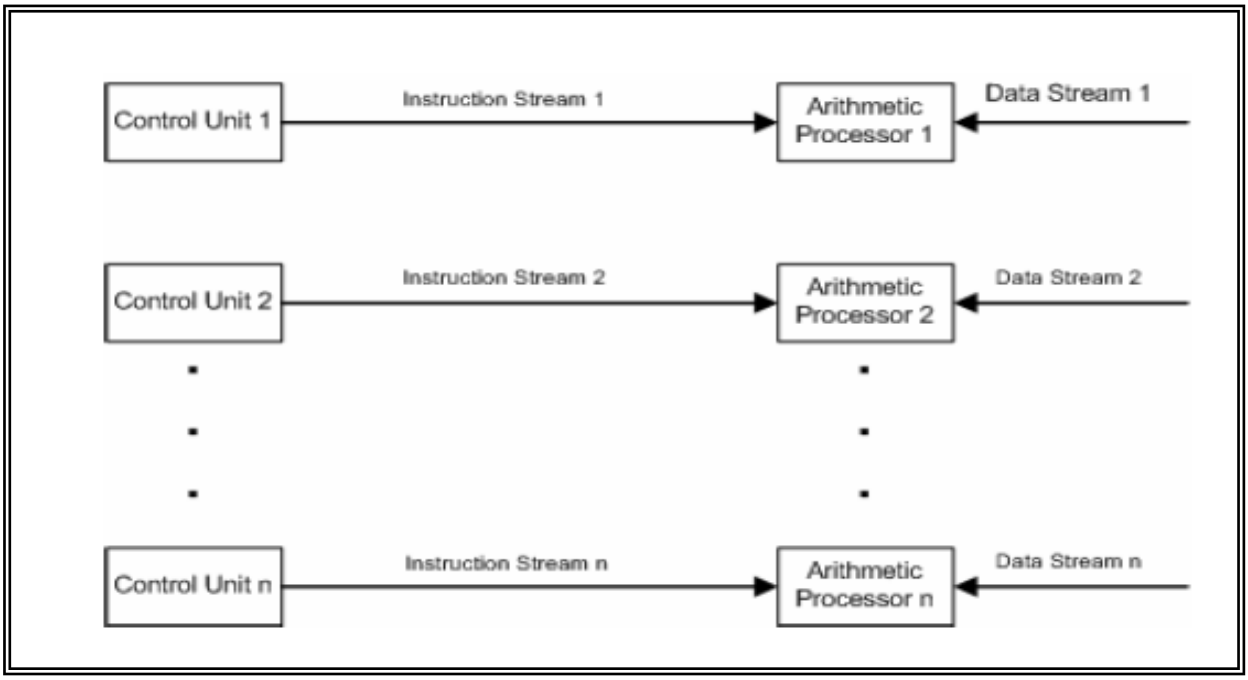


Figure 5.2: Model of a MIMD architecture.

5.4 Algorithm for Parallel Implementation of AES

There are two major components of parallel algorithm design. The first one is identification and specification of the overall problem as a set of tasks that can be performed concurrently. The second is the mapping of these tasks onto different processors so that the overall communication overhead is minimized. The first component specifies concurrency, and the second one specifies data locality. The performance of an algorithm on a parallel architecture depends on both concurrency and data locality. Concurrency is necessary to keep the processors busy. Locality is important because it minimizes communication overhead. Ideally, a parallel algorithm should have maximum concurrency and locality. However, for most algorithms, there is a tradeoff. An algorithm that has more concurrency often has less locality [52].

To implement the AES algorithm in parallel, data blocks and a key are distributed among the available processors. Each processor will encrypt different data blocks using the same key. For example, in order to encrypt **n** number of data blocks with p processors, n/p data blocks will be encrypted by each processor. As each processor has its own data blocks and a key (increases data locality), all the 10/12/14 rounds (consists of four transformations) will be executed by each processor for encrypting each data block [53].

After encrypting all the data blocks of each processor, the encrypted data will be merged (Figure 5.3) in tree structure and return back to the main processor. For example, if there are four processors working in parallel, processor P1 will send its encrypted data to P0 and P0 will merge its encrypted data with P1; processor P3 will

send its encrypted data to P2, and P2 will merge its encrypted data with P3. Finally processor P2 will send its (P2 & P3) encrypted data to P0 and P0 will merge its (P0 & P1) encrypted data with P2. This technique of merging and returning data to the main processor will increase the concurrency and reduce the idle time of each processor [54].

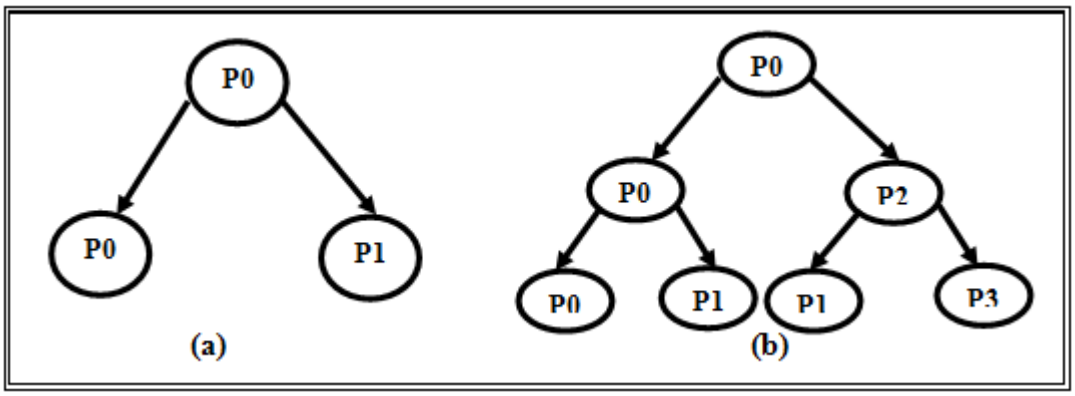


Figure. 5.3: (a) Data blocks are distributed between 2 processors.

(b) Data blocks are distributed among 4 processors.

5.5 Run Time Complexity of the Parallel Implementation

Time complexity is the most important measure of the performance of a parallel algorithm, since the primary motivation for parallel computation is to achieve a speedup in the computation. Parallel algorithms are executed by a set of processors and usually require inter-processor data transfers to complete execution successfully. The time complexity of a parallel algorithm to solve a problem of size n is a function T(n, p) which is the maximum time that elapses between the start of the algorithm's execution by one processor and its termination by one or more processors with regard to any arbitrary input. There are two different kinds of operations associated with parallel algorithms. These are the elementary operation and the data routing operation [55]. **Elementary operation** is an arithmetic or logical operation performed locally by a processor. **Data routing operations** refer to the routing of data among processors for exchanging the information. The time complexity of a parallel algorithm is determined by counting both elementary steps and data routing steps. A corollary follows that the time complexity of a parallel algorithm depends on the type of computational model being used as well as on the number of processors available.

Parallel computations are usually structured as a set of tasks executing concurrently and cooperatively on concurrent systems. Besides the actual service time spent in the system resources, execution time of a parallel computation consists of two kinds of additional delay: Queuing delay and Synchronization delay. Queuing delay results when two or more tasks compete for resources in the system. Synchronization delay results when a task has to idle and wait for others to finish before continuing. Because of the presence of queuing and synchronization delays, execution times of parallel computations are very difficult to predict. It is not possible to measure the performance of a parallel algorithm just by evaluating the run time complexity as shown in figure 5.4. It is also important to evaluate the speedup and efficiency of the algorithm [56]. All of these performance metrics are described below.

5.5.1 Parallel Run Time

The serial run time of a program is the time elapsed between the beginning and the end of its execution on a sequential computer. The parallel run time is the time that elapses from the moment that a parallel computation starts to the moment that the last processor finishes execution. The serial and parallel run time is denoted by TS and TP respectively. *The equation of parallel time is given as follows:*

Tp= Tcomp + Tcomm + Tsynch + Toverhead

Where Tcomp is the time of computation, Tcomm is the time of communication, Tsynch is the time of synchronization, Toverhead is the time of overhead communication.

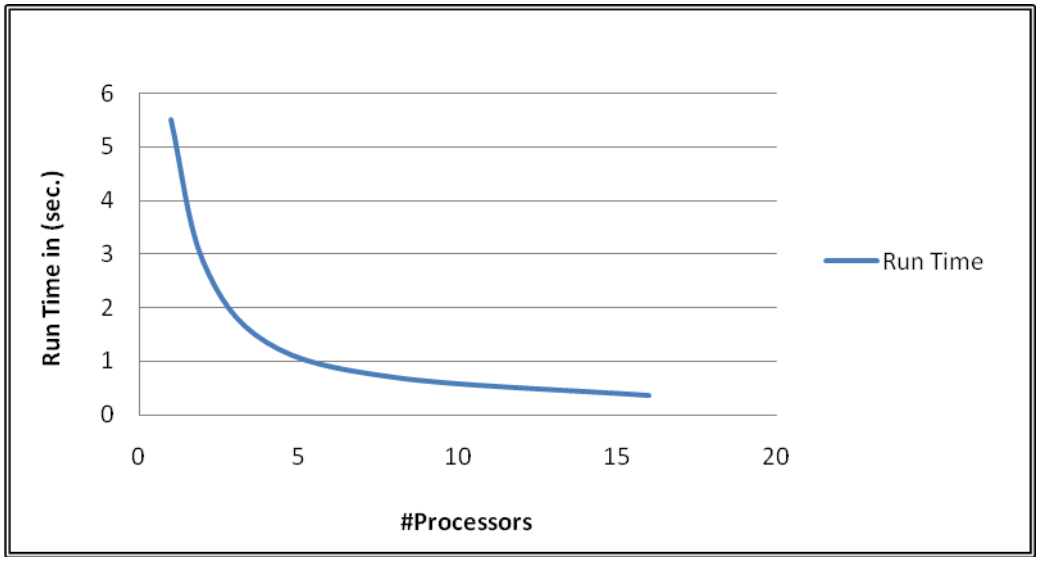


Figure 5.4: Run time as a function of number of processors.

5.5.2 Speed-up

When evaluating a parallel system, it is often important to know how much performance gain is achieved by parallelizing a given application over a sequential implementation. Speed-up is defined as the ratio of the time taken to solve a problem on a single processor to the time required to solve the same problem on a parallel computer with p identical processors. The speedup is denoted by the symbol S. Therefore,

S = TS / TP

Formally, the speedup S is defined as the ratio of the serial run time of the **best sequential algorithm** for solving a problem to the time taken by the parallel algorithm to solve the same problem on P processors. The speedup S has the upper and lower bounds as follows:

1<=S<=p

So, when the number of processors is equal to one, the speedup is one, but when number of processors is increased the speedup is increased. The speedup will never exceeds the number of processors in the parallel machine as shown in Figure 5.5.

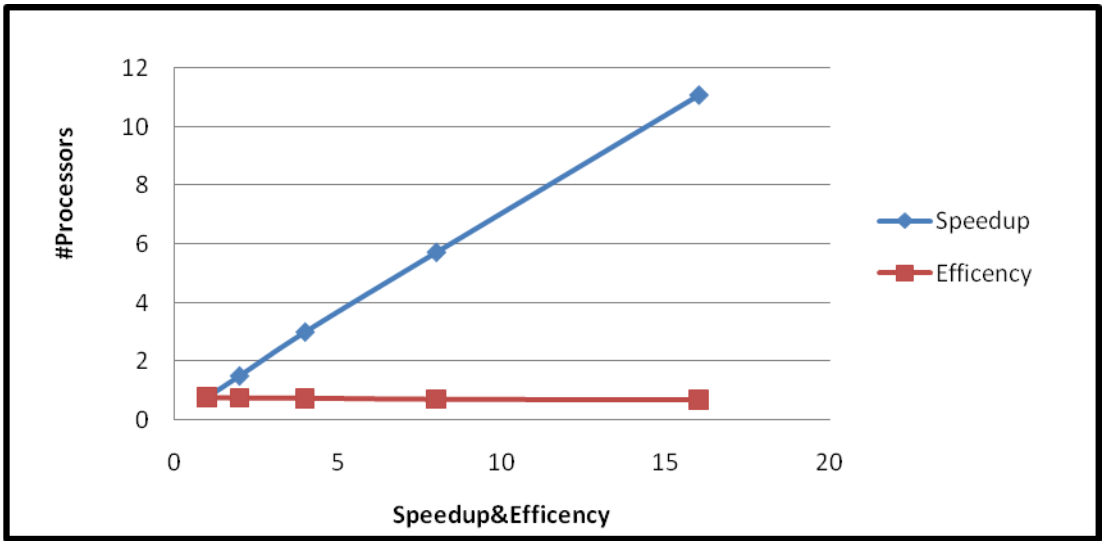


Figure 5.5: Speedup and Efficiency as a function of number of processors.

5.5.3 Efficiency

Efficiency is defined as the Speed-up with P processors divided by the number of processors P. Conceptually, the efficiency of the algorithm measures how well all P processors are being used when the algorithm is computed in parallel. An efficiency of 100 percent means that all of the processors are being fully used all the time. Efficiency is denoted by E. Therefore,

$$E = S / P$$
$$= TS / P\ TP$$

In an ideal parallel system, speedup is equal to number of processors P and efficiency is equal to 1. In practice, ideal behavior is not achieved because while executing a parallel algorithm, the processors cannot devote 100 percent of their time to the computations of the algorithm. Because, some part of the time is spent in inter-processor communication. The efficiency is decreased as number of processors are increased as shown in figure 5.5.

5.5.4 Overhead Communication

- *Sources of overhead communication in parallel programs:*
  - **Overheads:** wasted computation, communication, idling, contention.
    1. Inter-process interaction.
    2. Load imbalance.
    3. Dependencies.
- *Total parallel overhead.*
  - Total time collectively spent by all processors :
    1. Processor Elements = pTp. Where p is the number of processors.
    2. Time spent doing useful work (serial time) = TS.
    3. Overhead function**:** *TO = pTP-TS.*

Where TO is a general function which contains all kinds of overheads. The overhead is increased as the number of the processors is increased as shown in figure 5.6.

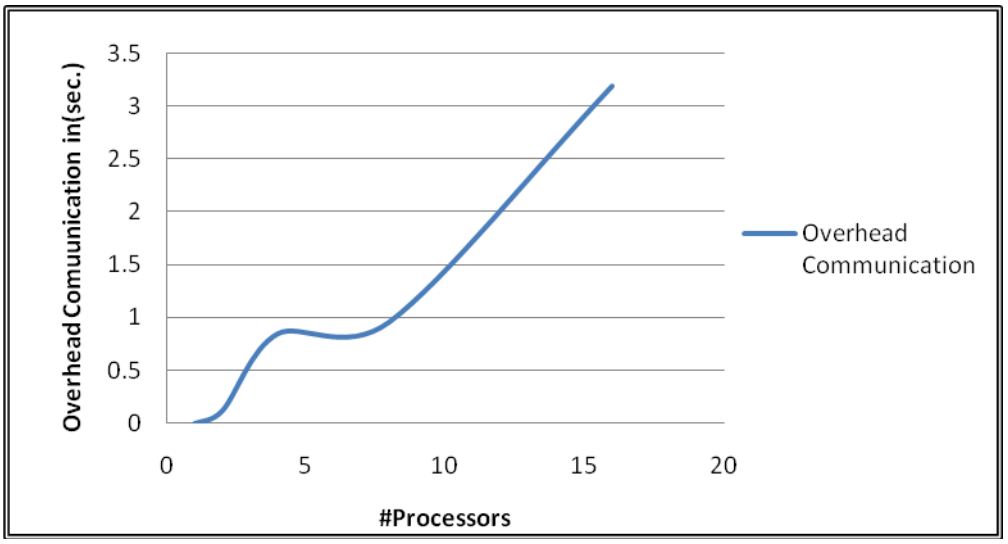


Figure 5.6: Overhead Communication as a function of number of processors.

5.6 Summary

There are two major components of parallel algorithm design. The first one is the identification and specification of the overall problem as a set of tasks that can be performed concurrently. The second is the mapping of these tasks onto different processors so that the overall communication overhead is minimized.

After implementing the AES algorithm in parallel, it is found that the performance of AES algorithm increases significantly as the number of processor increases. It is not possible to get the speedup to be equal to P (number of processors), as some parallel processing overhead is also incurred during the implementation of AES in parallel.

Chapter 6

Proposed AES

In this chapter we discuss our proposed enhanced sequential encryption algorithm. We also develop an original parallel AES based on the enhanced sequential AES.

The proposed enhanced sequential encryption algorithm increases the security of AES by combining password and time and using secure hash algorithm and message authentication code. The proposed parallel AES has high performance by using parallel implementation of the sequential counterpart. It increases performance through determining the task size of the data file as a percentage of the original data file size.

Introduction

The protection of data is the key mechanism for data security. The challenges faced to protect the data explore new encryption algorithms [57]. The evolution of encryption of encryptions in the field of cryptography may provide better security than single encryption routine.

The most important requirement for a new cryptographic algorithm is scalability. Implementers should be able to scale the algorithm from a bit-serial implementation to a highly parallel implementation depending on the desired maximum power consumption and speed.

The security of encrypted data depends on several factors like what algorithm is used for, what is the key size and how was the algorithm implemented in the product. Hence, how we can make a balance between the key size and the speed of the encrypted algorithm such that it has high speed and strong encryption.

Since most of complex encryption algorithms suffer from slow speed and in many times consume more power, which affects the encryption algorithm performance. In addition, how parallel computing will help us in improving our enhanced encryption algorithm and how will it affect developing the security of the database on the server side per database column to obtain overall security policy when encrypting addition data types (text, integer, image,……..) [58].

We develop the proposed enhanced AES and the parallel enhanced AES algorithms using JAVA with MPI (Message Passing Interface) support for parallel execution on distributed memory multiprocessors.

6.1 Tools Used in Algorithm Development

6.1.1 MPJ Express

MPJ Express is a message passing library that is used by the application developers to develop and execute parallel Java applications on compute clusters or network of computers. MPJ Express is originally designed for distributed memory machines like clusters but also supports efficient execution of parallel applications on desktops or laptops that contain shared memory or multi-core processors [59].

MPJ Express is a reference implementation of MPI Java 1.2 API, which is an MPI-like API for Java defined by the Java Grande Forum .*The current release contains:*

- The core library.
- The runtime infrastructure.
- The test-suite.

6.1.2 MPJ Express Configuration

MPJ Express can be configured in two ways: Multicore configuration, and cluster configuration.

Multicore Configuration: This configuration is used by developers who want to execute their parallel Java applications on multicore or shared memory machines (laptops and desktops) [59].

Cluster Configuration: This configuration is used by developers who want to execute their parallel Java applications on distributed memory platforms including clusters and network of computers [60].***A sample program with MPJ Express is*** *shown in Figure 6.1.*

```
import mpi.*;

public class HelloWorld {

public static void main(String args[]) throws Exception {
  MPI.Init(args);
  int me = MPI.COMM_WORLD.Rank();
  int size = MPI.COMM_WORLD.Size();
  System.out.println("Hi from <"+me+">");
  MPI.Finalize();
 }
}
```

Figure 6.1: MPJ Express Example.

6.2 Enhanced AES Algorithm

Advanced Encryption Standard consists of three block ciphers. They are: AES-128, AES-192, and AES-256. Each of the these standard ciphers is 128-bit block size with key sizes of 128, 192 and 256 bits respectively. To get top security we will use key sizes of 192 or 256, but this consumes a lot of processing time, and it is a low speed algorithm and consumes a lot of power. Key size of 128 bit gives us good security level and consumes less power than key sizes 192 and 256. We propose improving the AES algorithm which uses the key size of 128 bit through implementing some enhancements such as secure hash algorithm, message authentication code. This improves the

performance of the algorithm in regards to security. In addition, we propose a parallel version of the enhanced proposed AES through determining the parallel task size as a percentage of the original data file. By executing this parallel AES on a multiprocessor system we achieve good performance in regards to speed up.

6.3 Security Enhancement of sequential AES-128bit

We want to increase the security of AES by increasing the security of key and data block as shown in figure 6.2.

6.3.1 Increasing the security of a key

We want to increase the security of a key by using the following parts:

1. Using Time (the time of the current encryption which is taken after the program is worked).
2. Using secure hash algorithm to generate the key size 256bit to encrypt the data file.

6.3.1.1 Using Time

We want to make a combination between the time and the key size for the following reasons.

- The time Generation depends on the computer clock cycle, so it is very difficult to predict it.
- The prediction of the combination between time and key size is very complicated problem.
- The time of the current encryption will send as an encrypted data in the encrypted data file to the second party.

We encrypted the current encryption time by using the following:

1. We use the SBOX to change the format of the current encryption time.
2. We use the fourth round key of the key size 128bit to generate other 10 round keys to encrypt the current encryption time.
3. After encrypted the data blocks with key size 256 bit according to figure 6.3, We put the current encryption time after achieving steps 1 & 2 between the encrypted data blocks, since the data blocks are encrypted by the round key size 128 bit which generation from secure hash algorithm with key size 256 bit, and the current encryption time is encrypted by the fourth round key size 128 bit of the original key size 128 bit too.
4. The intruder will not have enough information to find the current encryption time for the following combined reasons:
   - The format of the current encryption time is changed.
   - The current encryption time is encrypted by the fourth round key of the key size 128 bit.

- The current encryption time is putted in dynamic position between the encrypted data blocks according to equations 6.3.1.1. So the half of the encrypted current time is merged with previous encrypted data block, and the second is merged with the next encrypted data block.

- The data blocks are encrypted by round key size 128 bit which generate from secure hash algorithm with key size 256 bit, and the current encryption time is encrypted by the fourth round keys 128 bit of the original one too.

- If the intruder catches the encrypted data file :

  - He should solve the equations 6.3.1.1, to determine the dynamic position of the encrypted current time.
  - He should know the original key size 128 bit to generate the fourth round key size 128 bit to decrypt the encrypted current time file which is not the suitable key to decrypt the data file, we want the new key size 256bit to decrypt the data file.

- The second party is the only one who knows how to extract the dynamic position of the current encryption time.

The following is the pseudo code of the current encrypted time module.

begin

j for columns (between blocks) //change its value every time the program is beginning

start

1. I for rows //change every getting new time
2. Position = #encrypted data file rows mod I // the position of the encrypted current time. ----------(Equation 6.3.1.1).
3. Put the encrypted current time in the coordinate (position ,j);
4. Send value = value(the random round key(1-10) size 128 bit of the original one) mod position// sending value to the second party.
   End

6.3.1.2 Using secure hash algorithm

After concatenation between the key size 128 bit and the current encryption time we will use secure hash algorithm to generate the new key size 256 bit. By this key we encrypted data blocks. We choose the secure hash algorithm for the following reasons:

1. It enjoys fast operation, since it doesn't need a key.
2. When the input is changed the output is changed directly.
3. It takes any arbitrary input data size and produce fixed data size.

The new generation key size will be 256 bit. We used 10 round keys instead of 14 round keys to encrypt the data file.

6.3.2 Increasing the security of the data blocks

We increase the security of data blocks by using message authentication algorithm (MAC) by using the new key size 256 bit as a key and data blocks, the size of each data block to be encrypted is 128 bit as shown in figure 6.3 part b .

Before making encryption we calculate the MAC value for each data block and store it in the encrypted output file. after making decryption we calculate the MAC value for each data block. We compare between these values if the value of any data block is not equal, we close the decrypted file since there is changing or attacking on it.

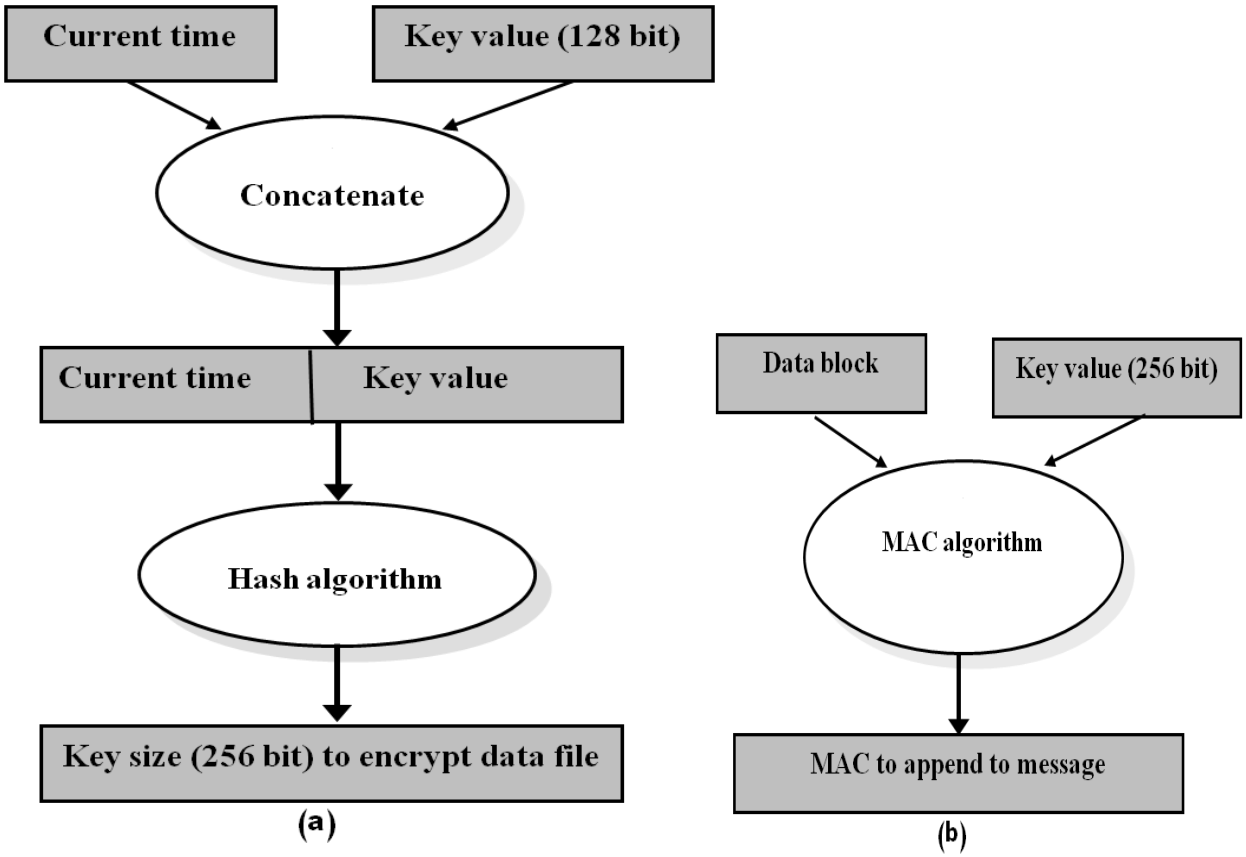


Figure 6.2: Sequential Enhanced Proposed AES Operations

6.3.3.1 Original Sequential Encryption Algorithm

The following is the pseudo code of the sequential original AES algorithm.

Algorithm Sequential_AES( )

Begin

1. Generate AES 16 bytes random key
2. Create cipher-1(Encrypt.mode, AES key)
3. while(is not last block) do

   3.1 Cipher text= Cipher-1(block-data)
4. End while loop
5. Write last block cipher to the output file
6. Write last block size to the output file

END

6.3.3.2 Proposed Enhanced Sequential Encryption Algorithm

We increase the security of the selected private key of the sequential AES by combining time and password and using the secure hash algorithm and message authentication code to guarantee the integrity of data. The following is the pseudo code of the enhanced sequential AES algorithm.

Algorithm Enhanced_Sequential_AES( )

Begin

1. Generate incremental I for columns // the column of the position of the encrypted current time
2. Input password, input file, output file
3. Initialize:CipherAES-128,DIGEST_ALG="SHA-256"
   MAC_ALG="HmacSHA256"
4. Generate key= password. get Bytes("UTF-16LE") //generate the
   // AES-128 key
5. Generate 8 bytes initial vector IV-1(8 current time)
6. Generate 16 bytes AES key-1 combination from IV-1 and key from password
7. Write IV-1 to output file
8. Create cipher-1(Encrypt. Mode, AES key-1, IV-1)
9. Create Hmac-1(AES key-1, HMAC_ALG)
10. Calculate a HMAC-SHA256 for previous cipher in line 1 and write it to out file

11. While(is not last block) do

    11.1 Cipher text= Cipher-1(block-data).

    11.2 MAC-data=Hmac-1(block-data). // make check for every block. So if one block of data is changed the encrypted file will not decrypt.

12. End while loop
13. Write last block cipher to output file
14. Write last block size to output file
15. Write last block MAC-data to output file
16. Call current-encrypted-time-module //determine the position of the encrypted current time.

END

6.4 Proposed Parallel Encryption algorithm

We use the proposed enhanced AES as module in our proposed parallel AES algorithm. We increase performance (speedup) by executing the proposed parallel AES algorithm in parallel on multiprocessors through reducing serialization by limiting use of barrier MPI statement which generates synchronization delay more than necessary, since. By limiting the number of barrier MPI statements there is less blocking asynchronous send and receive which allows a greater overlapping between computation and message passing communication. Also removal of the barrier MPI statement allows parallel processors to return control more quickly to the calling process. Writing the code for the parallel AES in Java with MPI library leads to appearance of many issues that must be handled in the MPI and MPJ Express package methods and other data distributing problems that must be solved.

6.4.1 Collective Communications

MPI provides a variety of routines for distributing and re-distributing data, gathering data, performing global sums etc. This class of routines comprises what are termed as the "collective communication" routines. Although a better term for this class of routines might be "collective operations". Figure 6.2 shows what distinguishes collective communication from point-to-point communication. Collective communication always involves *every* process in the specified communicator [61].

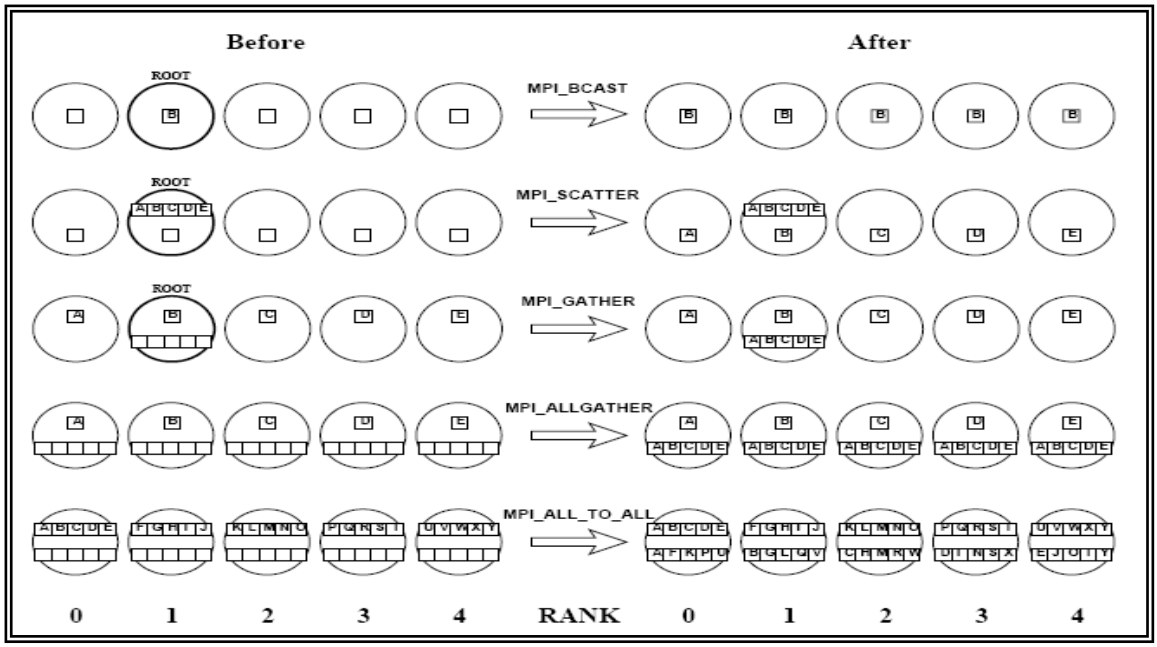


Figure 6.2: MPI Collective Communication.

Barrier synchronization: This is the simplest of all the collective operations. It involves no data at all. **MPI.COMM_WORLD.Barrier( )** blocks the calling process until all other group members have called it. As an example of using Barrier synchronization is that in one phase of a computation, all processes participate in writing a file. The file is to be used as input data for the next phase of the computation. Therefore no process should proceed to the second phase until all processes have completed phase one.

A broadcast MPI.COMM_WORLD.Bcast has a specified root process and every process receives one copy of the message from the root. All processes must specify the same root (and communicator).

MPI.COMM_WORLD,Gather: Each process sends the contents of its send buffer to the root process.

MPI.COMM_WORLD.Scatter : the inverse of Gather.

6.4.2 Scatter a file by MPJ

public void **Scatter** (java.lang.Object sendbuf, int sendoffset, int sendcount,

Datatype sendtype, java.lang.Object recvbuf, int recvoffset,

int recvcount, Datatype recvtype, int root)

throws MPIException.

6.4.3 The Parallel Encryption Algorithm

The following is the pseudo code of the proposed enhanced parallel encryption AES algorithm which uses the enhanced sequential AES as a called module.

Algorithm Parallel_AES_Encrypt_Algorithm(Input password, input file, output file)

Begin

1. myRank=MPI.Rank() //determine the rank of the process
2. size= MPI.Size()// total number of processes
3. call Prepare_Data_File(inFile)
4. call Enhanced_Sequential_AES_Encrypt() return result to encrypt byte array
5. allocate byte array alldata[encrupt_size*size]//dealing with bytes
6. MPI.Gather(encrypt[],alldata[],root)//gathering data from all other
7. processes and return back to the root process
8. If(my Rank==root)

   8.1 Write all data[] to output file

9. End if

END

The following is the pseudo code of the parallel data file preparation subroutine.

Subroutine Prepare_Data_File (Input: input file in File, Output: output file out File)

Begin

1. If(my Rank==root)
    1.1 Calculate size of in File
    1.2 Calculate work int. work=size/#processors
    1.3 Calculate remainder r=size mod #processores
    1.4 Store work and r in array worksize[]
2. End if
3. MPI.Bcast(worksize[],root)
4. Allocate myjob[work]
5. If(r!=0 and myRank=#p-1)
        5.1 Allocate myjob[work+r]
6. End if
7. MPI.Barrier
8. Scatter(inFile,work,root)
9. MPI.Barrier
10. If(myRank==root and r!=0)
    10.1 MPI.ISend(inFile, last r bytes,#p-1)
11. End if
12. If(myRank==#p-1 and r!=0)
    12.1 MPI.IRecv(myjob, last r location ,root)
13. End if

END

The following is the pseudo code of the proposed enhanced parallel decryption AES subroutine.

Subroutine Parallel_AES_Decryption(Input password, input file, output file)

Begin

1. myRank=MPI.Rank()
2. size= MPI.Size() // return the number of processes
3. call prepareFile(inFile)
4. call Deccrypt()// return result to decrypt byte array
5. allocate alldata[worksize[0]*#p-1+worsize[1]]
6. Gather(decrypt,worksize[0],root)
7. If(myRank==#p-1 and worksize[0]!=worksize[1])
    7.1 Diff=worksize[1]-worksize[0]
    7.2 MPI.ISend(decrypt, last diff bytes, to root)
8. End if
9. If(myRank==root and worksize[0]!=worksize[1])
    9.1 Diff=worksize[1]-worksize[0]
    9.2 MPI.IRecv(alldata, in last diff location, from last p)
10. End if
11. Write alldata[ ] to output file

END

The following is the pseudo code of the proposed enhanced parallel decryption AES subroutine.

```
Subroutine Parallel_AES_Decrypt( )
    Begin
    1. Allocate worksize[2]
    2. Call SequentialDecrypt() return result to decrypt byte array.
    3. If(myRank!=#p-1)
            3.1Worksize[0]=decrypt.length.
            3.1 If(myRank==0)
            3.2.1MPI.ISend(worksize[0], to #p-1)
            3.2 End if
    4.  End if
    5. If(myRank==#p-1)
            5.1 MPI.IRecv(worksize[0], from root)
            5.2 Worksize[1]=decrypt.length
    6. End if
    7. MPI.Bcast(worksize[1],from #p-1)
    END
```

Note that in the decryption process we used Prepare_File() procedure mentioned in the encryption method to make our system more friendly and more productive because in future work we will add a subsystem for searching the data files. When a process finds a target data file it begins as the root process machine. So it is not a good idea to make the root process fixed (e.g., the process with the maximum rank). In our solution this approach makes the parallel system more transparent and scalable.

6.5 Tests and results-Multiprocessor Systems

In the following measurement experiments we want to find the suitable parallel task size that achieves our goals of good load balance among parallel processors, small communication overhead, small parallel time, and best speedup and performance.

The results in this section are only applicable to the AES encryption algorithm test case. These results are application dependent. They may change as different applications as used to test our novel techniques. Hence, the optimal task ratio that has been measured (15-25%) may be different for different applications.

We test our proposed parallel AES using the following multiprocessor systems:

Processor Pentium 4	Clock Speed :2.8GHZ	Ram:1GB	Cache:256 KB
Processor Core2dueo	Clock Speed :2 GHZ	Ram:1.96GB	Cache:256 KB
Processor Core2Quad	Clock Speed :2.4GHZ	Ram:2GB	Cache:512 KB
Processor Core I8	Clock Speed :2 GHZ	Ram:1GB	Cache:256 KB

6.6 Experiments

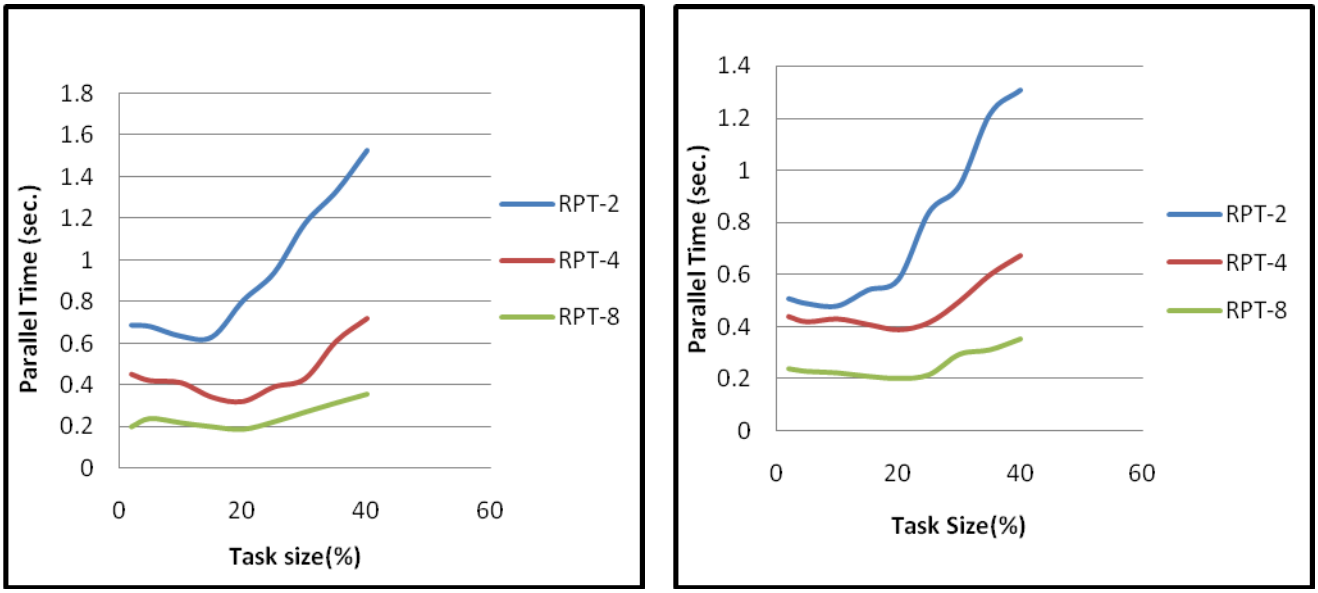
The relation between parallel task size different performance metrics(speedup, parallel run time, efficiency, and overhead communication) for different number of processors(2, 4 and 8) is shown in the following experiments.

Experiment 1: Relation between parallel task size and parallel run time for different data file sizes (25MB, 84MB and 144MB) and different number of processors (2, 4, and 8).

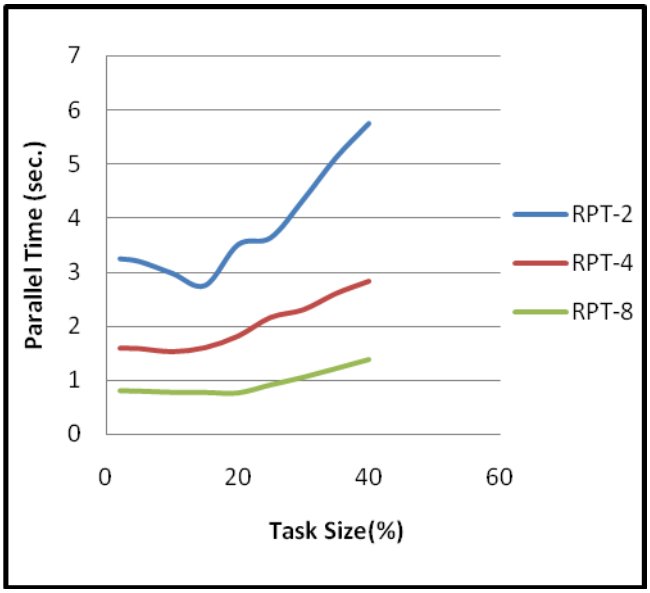
Figure 6.3 shows the effect of varying parallel task size on the parallel run time for different data file sizes for number of processors of 2, 4, and 8 in the multiprocessor system.

Figure 6.3 (a) shows the effect of varying parallel task size on the parallel run time for data file size of 25MB. Figure 6.3 (b) shows the effect of varying parallel task size on the parallel run time for data file size of 84MB. And Figure 6.3 (c) shows the effect of varying parallel task size on the parallel run time for data file size of 144MB.

As shown in figure. 6.3 the parallel task size between 15% and 25% of the total data file size gives the best performance (parallel run time). So, if the parallel task size is small we get good load balancing but the number of merging is increased. This leads to higher task management, more contention and more inter communication. So if the parallel task size is too small the number of merging increases and the communication increase, so the processors will consume their power in merging more than in processing the task size and hence, the run time will increase. If the task size is big, then this leads to load imbalance and some of processors will be idle and the parallel time will increase.



(a) **(b)**



(c)

Figure 6.3: Parallel run time of the proposed parallel AES (128 bit key) with 2,4 and 8 processor. (a) data file size of 25MB. (b) data file size of 84MB. (c) data file size of 144MB.

Experiment 2: Relation between parallel task size and speed up for different data file sizes (25MB, 84MB and 144MB) and different number of processors (2, 4, and 8).

Figure 6.4 shows the effect of varying parallel task size on the speedup for different data file sizes for number of processors of 2, 4, and 8 in the multiprocessor system.

Figure 6.4 (a) shows the effect of varying parallel task size on the speedup for data file size of 25MB. Figure 6.4 (b) shows the effect of varying parallel task size on the speedup for data file size of 84MB. And Figure 6.4 (c) shows the effect of varying parallel task size on the speedup for data file size of 144MB.

As shown in figure. 6.4 the parallel task size between 15% and 25% of the total data file size gives the best performance (speedup). So, the optimal task size gives us good load balance with little communication and high data locality and high speed up.

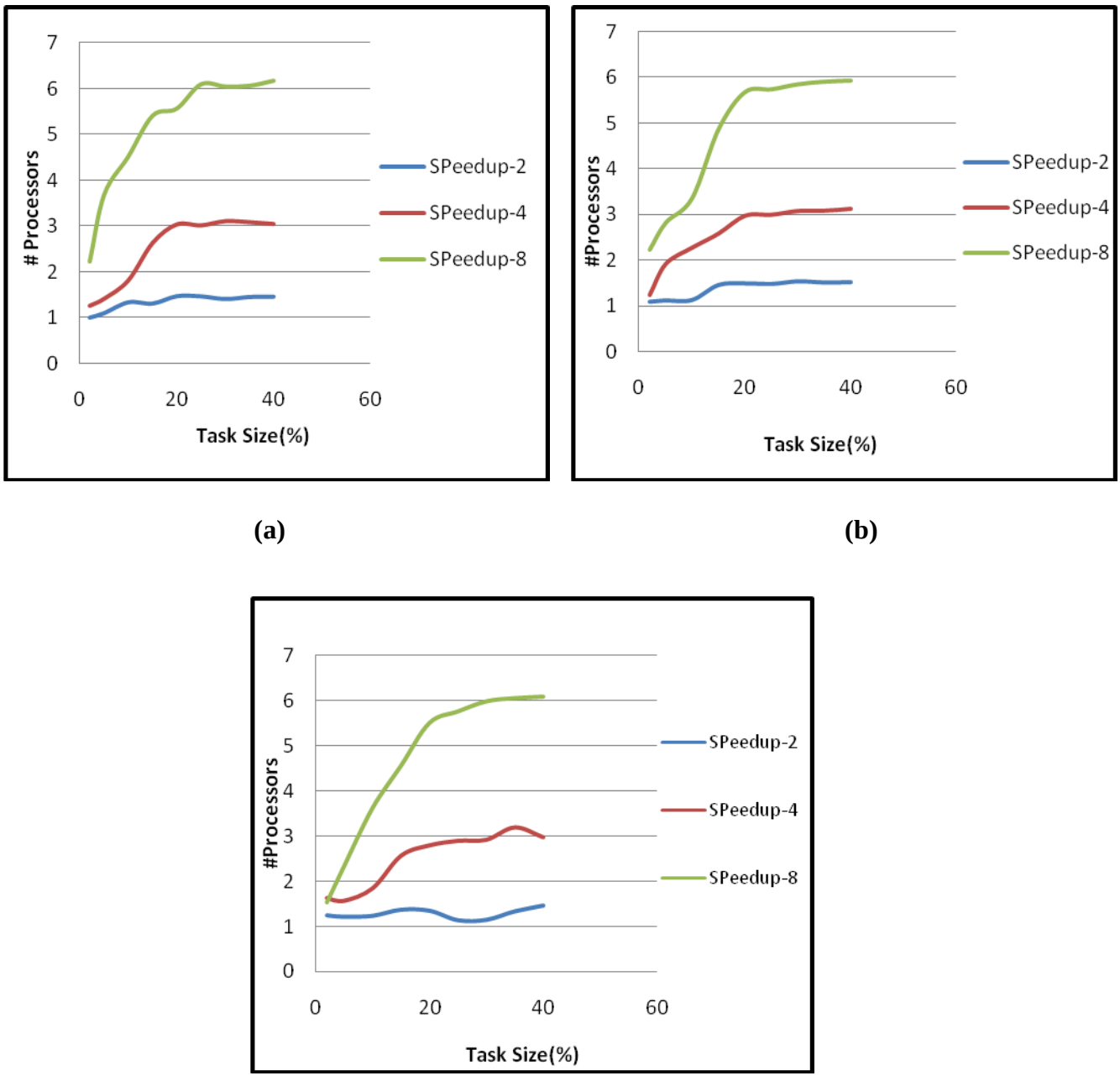


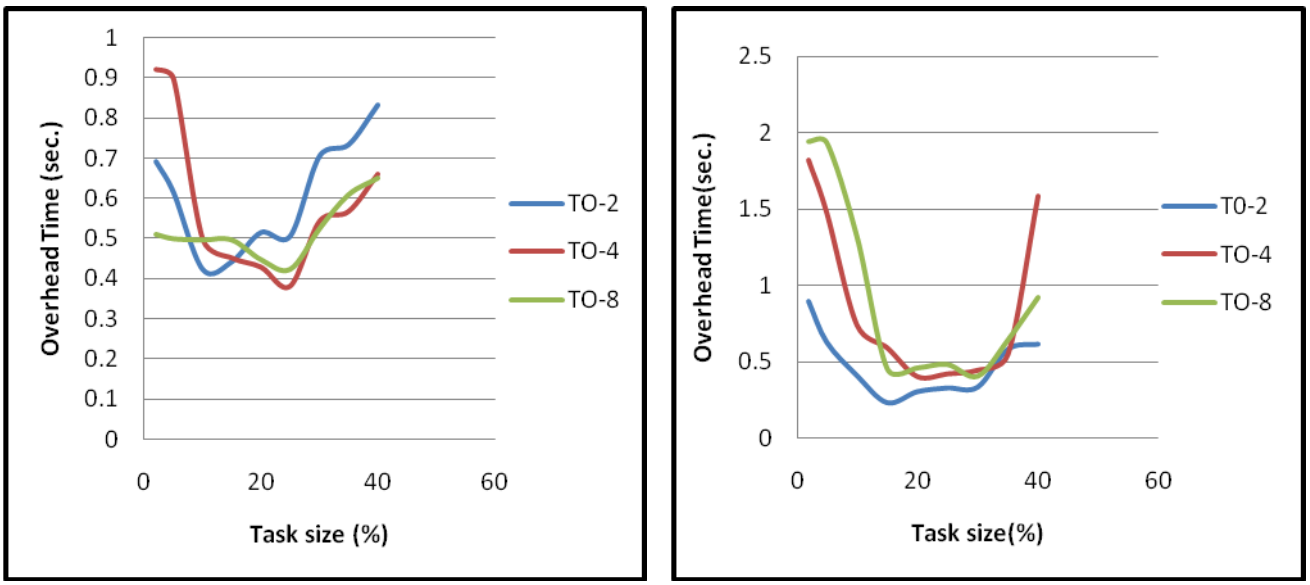
Figure 6.4: Speedup of the proposed parallel AES (128 bit key) with 2,4 and 8 processor. (a) data file size of 25MB. (b) data file size of 84MB. (c) data file size of 144MB.

Experiment 3: Relation between parallel task size and communication overhead for different data file sizes (25MB, 84MB and 144MB) and different number of processors (2, 4, and 8)

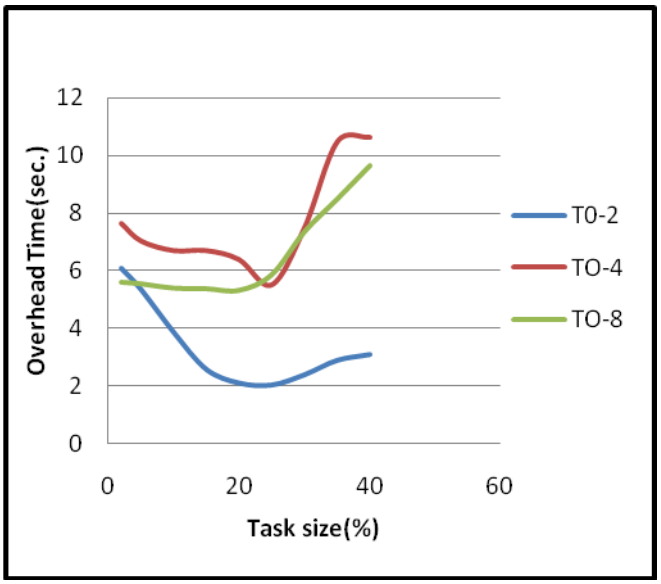
Figure 6.5 shows the effect of varying parallel task size on the communication overhead for different data file sizes for number of processors of 2, 4, and 8 in the multiprocessor system.

Figure 6.5 (a) shows the effect of varying parallel task size on the communication overhead for data file size of 25MB. Figure 6.5 (b) shows the effect of varying parallel task size on the communication overhead for data file size of 84MB. And Figure 6.5 (c) shows the effect of varying parallel task size on the communication overhead for data file size of 144MB.

As shown in figure. 6.5 as we increase the parallel task size the communication overhead starts large then it decreases as we increase parallel task size, then it increases again. The parallel task size between 15% and 25% of the total data file size gives the best performance (communication overhead). So, the optimal task size gives us good load balance with little communication and high data locality and small communication overhead. So, when the parallel task is too small there is more contention, more intercommunication and the processors consume more power in merging than in processing the tasks. When the parallel task size becomes bigger, we get good task size which gives us good load balancing with little communication and high data locality and lower overhead communication. When parallel task size becomes more bigger, the execution of the algorithm becomes more sequential and concurrency decrease.



(a) **(b)**



(c)

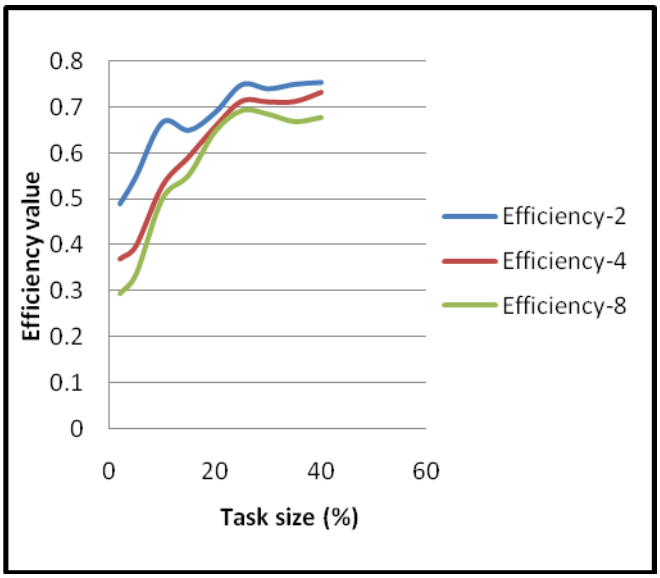
Figure 6.5: Communication overhead of the proposed parallel AES (128 bit key) with 2, 4 and 8 processor. (a) data file size of 25MB. (b) data file size of 84MB. (c) data file size of 144MB.

Experiment 4: Relation between parallel task size and efficiency for different data file sizes (25MB, 84MB and 144MB) and different number of processors (2, 4, and 8).

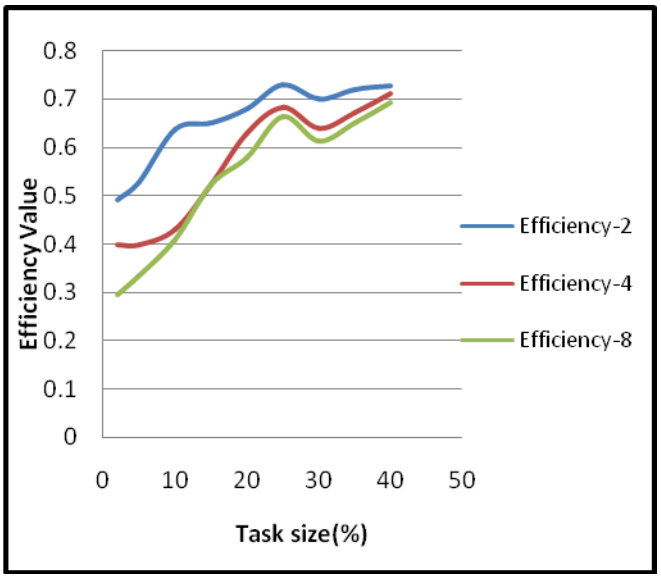
Figure 6.6 shows the effect of varying parallel task size on the efficiency for different data file sizes for number of processors of 2, 4, and 8 in the multiprocessor system.

Figure 6.6 (a) shows the effect of varying parallel task size on the efficiency for data file size of 25MB. Figure 6.6 (b) shows the effect of varying parallel task size on the efficiency for data file size of 84MB. And Figure 6.6 (c) shows the effect of varying parallel task size on the efficiency for data file size of 144MB.

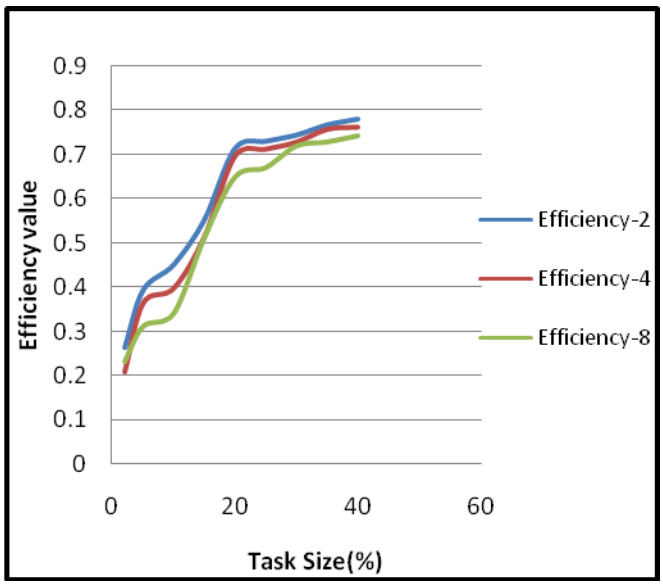
As shown in figure. 6.6 the parallel task size between 15% and 25% of the total data file size gives the best performance (efficiency). So, if the parallel task size is small we get good load balancing but the number of merging is increased. This leads to higher task management, more contention and more inter communication. So if the parallel task size is too small the number of merging increases and the communication increase, so the processors will consume their power in merging more than in processing the task size and hence, the run time will increase. If the task size is big, then this leads to lead to load imbalance and some of processors will be idle and the parallel time will increase.



(a)



(b)



(c)

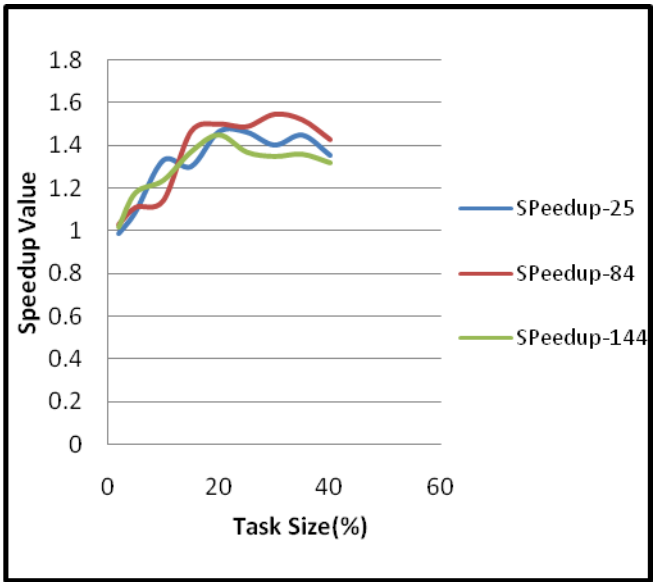
Figure 6.6: Efficiency of the proposed parallel AES (128 bit key) with 2, 4 and 8 processor. (a) data file size of 25MB. (b) data file size of 84MB. (c) data file size of 144MB.

Experiment 5: Relation between parallel task size and speed up for different data file sizes (25MB, 84MB and 144MB) and different number of processors (2, 4, and 8).

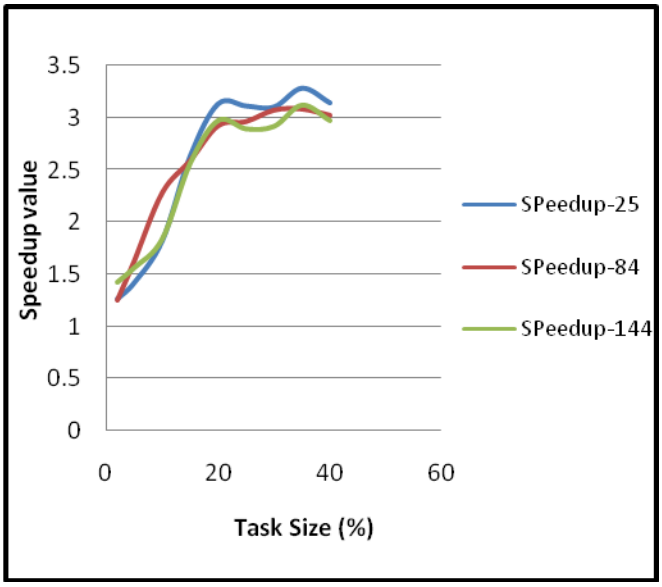
Figure 6.7 shows the effect of varying parallel task size on the speedup for different data file sizes for number of processors of 2, 4, and 8 in the multiprocessor system.

Figure 6.7 (a) shows the effect of varying parallel task size on the speedup for 2 processors. Figure 6.7 (b) shows the effect of varying parallel task size on the speedup for 4 processors. And Figure 6.7 (c) shows the effect of varying parallel task size on the speedup for 8 processors.

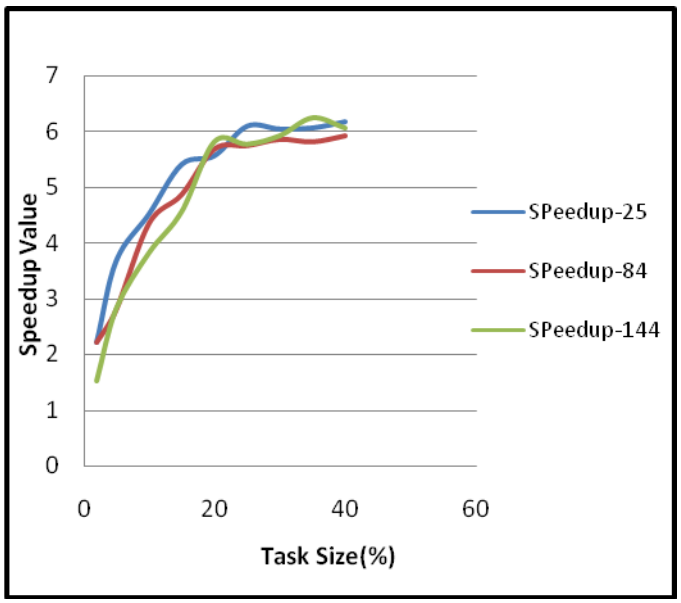
As shown in figure. 6.7 the parallel task size between 15% and 25% of the total data file size gives the best performance (speedup). So, the optimal task size gives us good load balance with little communication and high data locality and high speed up.



(a)



(b)



(c)

Figure 6.7: Speedup of the proposed parallel AES (128 bit key) with 2,4 and 8 processor. (a) 2 processors. (b) 4 processors. (c) 8 processors.

6.7 Summary of Experimental Results

The goal of our experimental analysis of the proposed parallel AES algorithm is to find the parallel task size that gives the optimal *performance by achieving the following:*

- Load balance (all processors are as busy as possible).
- Small communication overhead between processors.

So, if the task size is small we get good load balancing among the processors but the number of merging is increased, higher task management is obtained, and more contention and more inter processor communication is generated. So if the task size is too small the number of merging will be increased and the communication is increased, so the processors will consume their power in merging than in processing the task size and the parallel run time will increase. If the task size is big, then this leads to load imbalance and some of processors will be idle and the parallel time will increase.

To get good performance in parallel computing *we should reduce serialization by the following approaches:*

a. **Limiting the use of barrier:** Excessive serialization is the use of synchronization more than necessary.
b. **Mutual exclusion:** We reduce serialization by using separate locks for separate data items.

As shown in the results of the developed experiments we can determine the suitable parallel task size to be between 15% and 25% of the total data file size. This optimal parallel task size gives us the best performance with low parallel run time and low power consumption.

Chapter 7

Conclusion and Future work

In this chapter, section 7.1 provides the conclusion of the thesis, whilst the suggested future work is proposed in section 7.2.

7.1 Conclusion

Advanced Encryption Standard (AES) consists of three block ciphers. They are: AES-128, AES-192, and AES-256. Each of the above standard ciphers is 128-bit block size with key sizes of 128, and 192 & 256 bits respectively. For top security AES uses key sizes of 192 or 256, but this takes high processing, low speed and take more power consumption. Key size of 128 bit gives us security level and takes less power consumption than key sizes of 192 and 256. So by improving AES with key size of 128 bit by using secure hash algorithm, and message authentication code and determining the parallel task size as a percentage of the original data file by using parallel computing we can achieve good performance in speed with acceptable security.

Experimental results of the proposed algorithms demonstrate the efficiency of the proposed AES-128. We increased the security of the selected private key by combining time and password and using the secure hash and message authentication code algorithms to guarantee the integrity of data. We get good performance in parallel computing by reducing serialization through Limiting use of barrier MPI statements which produce more synchronization delay than necessary. Minimizing the use of barrier MPI statement reduces the blocking asynchronous send and receive and allows the greatest overlap between computation and message passing. It also does so by returning control most quickly to the calling process.

Our Proposed AES-128 parallel algorithm enjoys high video file resolution and quality of encryption. In fact, all experiments show that the system keeps the high video file resolution and the quality of the data file. We recommend our system for encryption and decryption of large video files because experiments show excellent speedup and guarantee quality.

It’s clear from the measurement results and visual inspection that the optimal parallel task size ranges from 15% to 25% of the data file size. This optimal parallel task size gives the best performance with low parallel run time and low power consumption.

7.2 Future Work

The work of the thesis may be extended by the following enhancements:

- **Using more key size than 128 bit to increase the security**

Encryption with AES is based on a secret key with 128, 192 or 256 bits. But if the key is easy to guess it doesn't matter if AES is secure, so it is as critically vital to use good and strong keys as it is to apply AES properly. Creating good and strong keys is a surprisingly difficult problem and requires careful design when done with a computer. Keys derived into a fixed length suitable for the encryption algorithm from passwords or pass phrases typed by a human will seldom correspond to 128 bits much less 256 bits.

- **Using hybrid algorithm of Advanced Encryption Standard and Elliptic Curve Cryptosystems**

The evolution of encryption of encryptions in the field of cryptography may provide better security than single encryption routine. So, by making a combination between the stronger public and private key algorithms we avoid the weakness of each of them. The most important requirement for a new cryptographic algorithm is scalability.

- **Using Cluster Configuration**

The cluster configuration is meant for users who plan to execute their parallel Java applications on distributed memory platforms including clusters or network of computers. Application developers can opt to use either of the two communication devices in the cluster configuration: *the communication devices including Java New I/O (NIO) device or Myrinet device as follows:*

1. Java New I/O (NIO) device driver known as niodev
2. Myrinet device driver known as mxdev

The Java NIO device driver (also known as niodev) can be used to execute MPJ Express programs on clusters or network of computers. The niodev device driver uses Ethernet-based interconnect for message passing.

- **Dynamic root depends on data locality**

MPJ express gives developers high scalability by converting traditional Ethernet network to cluster, of course there are limitation on load balancing, so in future work we hope to build a subsystem to determine the distributing of data over network to decide the rank of root according the locality of data.

References

[1] A. Menezes, P. van Oorschotand, and S. Vanstone, **"Handbook of Applied Cryptography,"**CRC Press, pp. 4 4- 15, 516, 1996.

[2] H. Delfs, and H. Kneble, **"Introduction to Cryptography Principles and Applications**", Springer, 2007.

[3] W. Stallings, "**Cryptography and network security Principles and Practices**", 4th Ed, Prentice-Hall, 2003.

[4] National Bureau of Standards, **"Data Encryption Standard"**, Federal Information Processing Standard (FIPS), Publication 46, National Bureau of standards, U.S. Department of commerce ,1997.

[5] M, Robshaw, **"Stream Cipher"**, RSA Laboratories, Technical Report TR-701,Version 2.0, July 1995.

[6] J. D. Golic, "**Cryptanalysis of alleged A5 stream cipher,"** in Advances in Cryptology-EUROCRYPT97, New York, Springer-Verlag, vol. LNCS 1233, PP 239-255,1997.

[7] M. Sharbaf, **"Quantum Cryptography: A New Generation of Information Technology Security System",** in Proc. The 6th international Conf. of information Technology: New Generations, Las Vegas, Nevada, pp. 1644-1648,2009.

[8] **http://electronicsbus.com/tutorials/search/AES+Encryption+Algorithm.**

[9] J. Buchmann, **"An Introduction to Cryptography"**, 2nd edition ,Springer, 2000.

[10] J. Daemon, and V. Rijmen, **"Rijndael: The Advanced Encryption Standard."** , Dr. Dobb's Journal, **26**, 3, March 2001, 137-139.

[11]**http://electronicsbus.com/tutorials/transient-key-cryptography-digital-signature.**

[12] W. Diffie, and M. Hellman, **"Multiuser Cryptographic Techniques",** proceedings of AFIPS National Computer Conference, 1976, 109-112 .

[13] J. Daemon, and V. Rijmen, **"The Rijndael Block Cipher: AES Proposal"**, NIST, Version 2, March 1999.

[14] **http://electronicsbus.com/tutorials/asymmetric-encryption-algorithms.**

[15] B. Schneier, **"Applied Cryptgraphy."**, New York: Wiley, 1996.

[16] National Policy on the Use of the Advanced Encryption Standard (AES) to Protect National Security Systems and National Security Information, CNSS Policy No. 1, Fact Sheet No. 1, June 2003, **www.nstissc.gov/Assets/pdf/fact%20sheet.pdf**

[17] Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, and Niels Ferguson. In Proc. 2nd AES candidate conference, pp 15–34,NIST, 1999, **www.macfergus.com/pub/icrijndael.html .**

[18] W. Stallings, **"Cryptography and Network Security: Principles and Practices."** Third Edition, Pearson Education, Inc. 2003.

[19] V. Rijmen, "The block cipher Rijndael",http://www.esat.kuleuven.ac.be/~rijmen/ rijndael/, (2001).

[20] M. Flynn, **"Some Computer Organizations and their effectiveness.",** IEEE Transactions on Computer, C-21, 9, 948-960, 1972

[21] National Institute of Standards and Technology: Data Encryption Standard (DES).**http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf, (2001).**

[22] **http://electronicsbus.com/tutorials/symmetric-key-encryption-algorithms.**

[23] **http://electronicsbus.com/tutorials/search/Generating+Digital+Signature/feed/rss2/** .

[24] **http://electronicsbus.com/tutorials/public-key-cryptographic-system/.**

[25] NIST (National Institute of Standards and Technology) Special Publication 800-57(May2006)**http://csrc.nist.gov/publications/nistpubs/800-57/SP800-57-Part1.pdf .**

[26] **http://www.ecrypt.eu.org/documents/D.SPA.21-1.1.pdf .**

[27] **http://electronicsbus.com/tutorials/message-authentication-codes-mac-cryptography-data-security/.**

[28] D. Coppersmith, **"The Data Encryption Standard (DES) and Its Strength Against Attacks.",** IBM Journal of Research and Development, May 1994.

[29] **A Performance Comparison of the Five AES Finalists**, 7 April 2000.

[30] A Thesis Presented in **"Partial Fulfillment of the Requirements for the Degree Master of Science in Computer Science"**, INDEPENDENT UNIVERSITY, BANGLADESH May 27, 2004.

[31] E. Biham, **"Design Tradeoffs of the AES Candidates,"** invited talk presented at ASIACRYPT'98, Beijing, 1998.

[32] About AES – **"Advanced Encryption Standard A short introduction 2007-08-24",** Copyright 2007, Svante Seleborg Axantum Software AB.

[33] P. Preneel, V. Rijmen, and A. Bosselaers, **"Principle and Performance of Cryptographic Algorithms",** Dr. Dobb's Journal, v.23, n. 12, 1998, pp. 126-131.

[34] M. Petkac and L. Badger, " **Security Agility in Response to Intrusion Detection** " Proceedings of the 16th Annual Computer Security Applications Conference (ACSAC) , New Orleans, LA, 2000, pp. 11–20.

[35] Blömer J. & Krummel, V, "**Fault Based Collision Attacks on AES,**" FDTC 2006, pp. 106-120.

[36] **http://www.cryptographyworld.com .**

[37] M. Abe, R. Gennaro, and K. Kurosawa, Tag-KEM/DEM: **"A new framework for hybrid encryption"**. J. Cryptol.21(1), 97–130 (2008).

[38] R. Cramer, and V. Shoup, **Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack**, SIAM J. Comput. **33**, 167–226 (2003)

[39] D. Hofheinz, and E. Kiltz, **"Secure hybrid encryption from weakened key encapsulation"**, in CRYPTO 2007, Springer LNCS, vol. 4622 (Springer, Berlin, 2007), pp. 553–571.

[40] K. Kurosawa, and Y. Desmedt, **"A New Paradigm of Hybrid Encryption Scheme"**, in CRYPTO'04. Springer LNCS, vol. 3152 (Springer, Berlin, 2004), pp. 426–442.

[41] **http://www.tech-faq.com/.**

[42] **http://en.wikipedia.org/wiki/Security/AES.**

[43] **http://en.wikipedia.org/wiki/Security/Round Keys.**

[44] L. Knudsen, and W. Meier, **"Correlations in RC6",** Fast Software Encryption, 7th International Workshop, Springer-Verlag, 2008.

[45] **Writing Message Passing Parallel Programs with MPI: A Two Day Course on MPI Usage**, Neil MacDonald, Elspeth Minty, Joel Malard, Tim Harding, Simon Brown, Mario Antonioletti.

[46] **"Introduction to Parallel Computing"**. Ananth Grama, Anshul Gupta, GeorgeKarypis,VipinKumar.**http://www-users.cs.umn.edu/~karypis/parbook/ .**

[47] C. Bell, **"Ultracomputer: A teraflop before its time"**, Communication of the ACM, 35, 8, 27-47, 1992.

[48] M. A tighetchi , P. Pal , F. Webber , R. Schantz , C. Jones , and J. Loyall , **" Adaptive Cyber Defense for Survival and Intrusion Tolerance** , " Internet Computing, IEEE , 8 : 25 – 33 , 2004 .

[49] I.Foster,**"Designing and Building Parallel Programs",http://www-unix.mcs.anl.gov/dbpp/ .**

[50] M. Flynn, **"Some Computer Organizations and their effectiveness"**, IEEE Transactions on Computer, C-21, 9, 948-960, 1972.

[51] J. van der Steen, and J. Dongarra, **"Overview of Recent Supercomputers"**, **www.phys.uu.nl/~steen/web03/overview.html.**

[52] M. Abbadi, **"A new message authentication technique using zigzag manipulation and block chaining"**, Applied Sci., 8: 3863-3870, 2008.

[53] **http://electronicsbus.com/tutorials/public-key-cryptographic-system/.**

[54] Wu, K; Karri, R.; Kuznetsov, G. & Goessel, M., "**Low Cost Concurrent Error Detection for the Advanced Encryption Standard**", International Test Conference, 2004. pp 1242- 1248.

[55] Ananth Grama, Anshul Gupta, GeorgeKarypis, and VipinKumar, **"Introduction to Parallel Computing"http://www-users.cs.umn.edu/~karypis/parbook/ .**

[56] **http://www.esat.kuleuven.ac.be/~rijmen/rijndael/ .**

[57] **http://www.truecrypt.org.**

[58] **http://www.crypto.com/papers/keylength.pdf.**

[59] **http://mpj-express.org/ .**

[60] **http://www.tech-faq.com/ .**

[61] Di Natale,G.; Flottes M.L. & Rouzeyre, B., **"An On-Line Fault Detection Scheme for SBoxes in Secure Circuits"**, Proc. of 13th IEEE International On-Line Testing Symposium, IOLTS 2007, pp. 57-62.

[62] **http://www.javamex.com/tutorials/cryptography/ciphers.shtml.**

[63] A. K. Lenstra, and E. R. Verheul, **"Selecting Cryptographic Key Sizes", Journal of Cryptology 14(4):255-293,** 2001.

[64] N. Ferguson, and B. Schneier, **"Practical Cryptography"**, Wiley, 2003.

[65] NIST, **Recommendation for Key Management: Part 1**, NIST. 2006.

[66] L. Chi-Feng, K. Yan-Shun, C. Hsia-Ling, and Y. Chung-Huang, **'Fast implementation of AES cryptographic algorithms in smart cards'**, IEEE 37th Annual 2003 International Carnahan Conference On Security Technology, pp. 573 – 579, 2003.

[67] M. Liberatori, F. Otero, J. Bonadero, and J. Castineira, 2007, **'AES-128 Cipher. High Speed, Low Cost FPGA Implementation'**, 3rd Southern Conference on Programmable Logic, pp. 195 – 198.