NEAR EAST UNIVERSITY

GRADEUATE SCHOOL OF APPLIED AND SOCIAL SCIENCES

ADVANCE ENCRYPTION STANDARD ANALYSIS AND IMPLEMENTATION

TAMER S. FATAYER

MASTER THESIS

DEPARTMENT OF COMPUTER ENGINEERING

Nicosia 2004

NEU **JURY REPORT**

DEPARTMENT OF COMPUTER ENGINEERING

Academic Year: 2003-2004

STUDENT INFORMATION

Full Name	Tamer S A Fatayer		
Undergraduate degree	BSc.	**Date Received**	**Spring** 1998-2002
University	The Islamic University of Gaza	**CGPA**	3.18

THESIS

Title	Advance Encryption standard Analysis and Implementation

Description
This thesis analyze the five candidate's algorithms for Advance Encryption Standard algorithm, specially analyze to RC6™. And to apply this algorithm as software over network communication channel to secure the data from the attacker

Supervisor	Prof. Dr. Fakhraddin Mamedov	**Department**	Computer Engineering

DECISION OF EXAMINING COMMITTEE

The jury has decided to accept / ~~reject~~ the student's thesis.
The decision was taken unanimously / ~~by majority~~.

COMMITTEE MEMBERS

Number Attending	3	**Date**	5/2/2004

Name	**Signature**
Assoc. Prof. Dr. Rahib Abiyev, Chairman of the jury	
Assist. Prof. Dr. Doğan Haktanır, Member	
Assoc. Prof. Dr. Ilham Huseynov, Member	

APPROVALS

Date
5/2/2004

Chairman of Department
Assoc. Prof. Dr. Doğan İbrahim

DEPARTMENT OF COMPUTER ENGINEERING
DEPARTMENTAL DECISION

Date:5/2/2004

Subject: Completion of M.Sc. Thesis

Participants: Prof. Dr. Fakhraddin Mamedov, Assoc. Prof. Dr. Rahib Abiyev, Assist.Prof. Dr. Doğan Haktanir, Assoc.Prof. Dr. Ilham Huseynov, Mohammed abdelal, Mohammed Aldiri.

DECISION

We certify that the student whose number and name are given below, has fulfilled all the requirements for a M .S. degree in Computer Engineering.

		CGPA
20021298	Tamer S A Fatayer	3.92

Assoc. Prof. Dr. Rahib Abiyev, Committee Member, Computer Engineering Department, NEU

Assist. Prof. Dr. Doğan Haktanir, Committee Member , Electrical and Electronic Engineering Department, NEU

Assoc. Prof. Dr. Ilham Huseynov, Committee Member, Computer Information System Department, NEU

Prof. Dr. Fakhraddin Mamedov, Supervisor, Dean of Engineering Faculties, NEU

Chairman of Department
Assoc. Prof. Dr. Doğan İbrahim

Tamer S A Fatayer : **Advance Encryption standard Analysis and Implementation.**

Approval of the Graduate School of Applied and Social Sciences

Prof. Dr. Fakhraddin Mamedov
Director

We certify this thesis is satisfactory for the award of the Degree of Master of Science in Computer Engineering

Examining Committee in charge:

Assoc. Prof. Dr. Rahib Abiyev, Member, Computer Engineering Department, NEU

Assist. Prof. Dr. Doğan Haktanir, Member , Electrical and Electronic Engineering Department, NEU

Assoc. Prof. Dr. Ilham Huseynov, Member, Computer Information System Department, NEU

Prof. Dr. Fakhraddin Mamedov, Supervisor, Dean of Engineering Faculties, NEU

ACKNOWLEDGEMENTS

Many people supported me during the completion of this thesis with criticism, helpful assistance and references. This thesis would have never been possible without them.

First, I would like to thank my supervisor the professor Fakhraddin Mamedov for his guidance and encouragement. He was a wonderful supervisor whose assistance and motivation were greatly appreciated.

I would like also to thank my brother Ali, and all my friends, especially Alaa El-Azayza, Hazem El-Baz, Mohammad klaib, Mohammad abu-Agrab, Hossam Salha, and Mohammad Basher for sharing wonderful comments.

I would like to thank my sisters heba, Myson, Myada, Anwar, and Khulud fatayer for them inspirational and moral support.

Last but certainly not least, I would like to show my gratitude to my parents Saad and Halema Fatayer, who have been generous with their encouragement throughout. This thesis is dedicated to them.

ABSTRACT

In the late 1990's DES was felt by many experts to be vulnerable to brute force attack by specialized computing machines. The National Institute of Standards and Technology held a public competition for replacement DES, and a number of algorithms were proposed.

Among the 15 preliminary candidates, MARS, RC6, Rijndeal, Serpent, and Twofish were announced as the finalist candidates on August 9, 1999 for further evaluation, which all these algorithms consider as symmetric-key encryption algorithms.

The five finalists will be the subject of further study before the selection of one or more of these algorithms for inclusion in the Advanced Encryption Standard (AES). A thesis discusses security, cryptography, public key, private key and many related topics.

Also a thesis discusses secure communication using Advanced Encryption Standard algorithm based on Rijndael, Serpent, Towfish, RC6™, and MARS algorithms. Also author developed a software using Visual C++ 6 for advanced RC6™. Also a thesis proves that all five of the finalist algorithms very strong cipher algorithms.

TABLE OF CONTENTS

INTRODUCTION

Information is becoming a crucial if not the most important resource of the economy and the society at large. Information differs radically from other resources, for instance, it can be copied without cost, it can be communicated at the speed of light, and it can be destroyed without leaving traces. This poses new challenges for the protection of this new resource and of intellectual property in general.

Information security, in particular cryptography, is an enabling technology that is vital for the development of the information society [28].

A cryptographic algorithm is the mathematical function used for encryption and decryption. There are two basic kinds of encryption algorithms are use today:

- Private key cryptography, which uses the same key to encrypt and decrypt the message. This type is also known as symmetric key cryptography.

- Public key cryptography, which uses a public key to encrypt the message and a private key to decrypt it [26].

This thesis discusses information security and cryptography, both types of cryptosystem and many related topics, but concerns on the symmetric key cryptography algorithms specially five candidates algorithms for AES theses algorithms are Rijndeal, Serpent, Twofish, RC6™, and Mars algorithms. It also concerns on the RC6™ algorithm and the related functions.

Author presents an application of symmetric key cryptography using the RC6™ to secure data over network and the Internet channels, with some modification on RC6™. Author adds another key to this algorithm, which makes it more secure.

The aim of this thesis is to analyze the five candidate's algorithms for Advance Encryption Standard algorithm, specially analyze to RC6™ which author make multiples encryption to each block to make the algorithm more secure. And to apply this algorithm as software over network communication channel to secure the data from the attacker.

This thesis includes four chapters covering the main topics related in the following structure:

Chapter 1, describes the overview to security and cryptography; which include information security, relation between security and cryptography, basic definitions of cryptosystem, main operation of cryptography, goal of cryptography, cryptanalysis, brut-force attack, and cryptology.

Chapter 2, describes cryptography algorithms, which include restricted algorithm Key based algorithms, operation is often used in symmetric algorithms, classical symmetric algorithms, Recent symmetric algorithms, Asymmetric cryptography algorithms, operation is often used in asymmetric algorithms, Common public key algorithms, One-time pads, Drawback of one-time pads.

Chapter 3, describes five candidates algorithms for AES, which include Rijndeal, Serpent, Twofish, RC6™, and MARS algorithms. And describe each algorithm how its work, encryption and decryption, and security. At end of this chapter there is comparison between these algorithms.

Chapter 4, presents an application of symmetric key cryptography developed by author using the RC6™ to secure data over network and the Internet channels. With some modification on RC6™ author will add another key to this algorithm which maybe consider as another private key to make the algorithm more secure or make it as public key to make this algorithms as asymmetric key cryptosystem.

Finally in conclusion the obtained important results for the thesis are given.

1. OVERVIEW OF SECURITY AND CRYPTOGRAPHY

Security may define as a condition that result from the establishment and maintenance of protective measures that ensures a state of inviolability from hostile acts or influences. With respect to classified matter, the condition that prevents unauthorized persons from having access to official information that is safeguarded in the interests of national security

Information systems professionals as being vital have long viewed security. Computing facilities and the information systems they support have become increasingly accessible as a result of the explosion of the open, public Internet since about 1993.

The demand for security safeguards has long been dominated by the military. As a result, the orientation is rather different from what corporations, government agencies and the public really need. Meanwhile, computing and communications specialists have dominated the supply of security safeguards. As a result, the language used is arcane.

Information is becoming a crucial if not the most important resource of the economy and the society at large. Information differs radically from other resources, for instance, it can be copied without cost, it can be communicated at the speed of light, and it can be destroyed without leaving traces. This poses new challenges for the protection of this new resource and of intellectual property in general. Information security, in particular cryptography, is an enabling technology that is vital for the development of the information society [28].

The concept of information will be taken to be an understood quantity. 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 [1].

1.1. Objectives associated with information security

- **Privacy or confidentiality:** is protection of transmitted data from passive attacks. With respect to release of message contents, several levels of protection can be id Keeps information secret from all but those who are authorized to see it [38].
- **Data integrity:** integrity can apply to a stream of messages, or selected fields within a message. Integrity to ensure the information has not been altered by unauthorized or unknown means.
- **Entity authentication or identification:** the authentication service is concerned with assuring a communication authentic, the function of authentication service is to assure the recipient that the message is from the source that is claims to be from. In this case of an ongoing interaction such as the connection of terminal to a host.
- **Signature** is a means to bind information to an entity.
- **Authorization:** Conveyance, to another entity, of official sanction to do or be something. Or Authorization is finding out if the person, once identified, is permitted to have the resource. This is usually determined by finding out if that person is a part of a particular group, if that person has paid admission, or has a particular level of security clearance.
- **Validation:** is a means to provide timeliness of authorization to use or manipulate information or resources.
- **Access control:** is the ability to limit and control the access to host system and application via communication links. To achieve this control, each entity trying to gain access must be identified, or authenticated, so that access rights can be tailored to the individual.
- **Certification:** Comprehensive evaluation of the technical and non-technical security features of an AIS (automated information system) and other safeguards, made in support of the approval/accreditation process, to establish the extent to which a particular design and implementation meet a set of specified security requirements Endorse of information by a trusted entity.
- **Timestamping**: A collection of digital data created with a purpose to prove temporal relationship (before, after) between different events. Records the time of creation or existence of information. In distributed databases, a concurrency control mechanism that assigns a globally unique timestamp to each transaction.

- **Witnessing:** verifies the creation or existence of information by an entity other than the creator.
- **Receipt:** receipts acknowledgement that information has been received.
- **Confirmation:** is acknowledgement that services have been provided.
- **Ownership:** means to provide an entity with the legal right to use or transfer a resource to anther. Or All rights, benefits and privileges under policies that are controlled by their owners. Policy owners may or may not be the insured. Ownership may be assigned or transferred by written request of current owner.
- **Anonymity:** conceals the identity of an entity involved in some process. Or Protection of names and other pieces of information that can identify participant's researchers do not ask participants to reveal information that would aid the researcher in identifying participant's individual data.
- **Non-repudiation:** prevents either sender or receiver from denying a transmitted message. Thus, when message is send, the receiver can prove that the massage was in fact sent by alleged sender. Similarly, when a message is receive, the sender can prove that the massage was in fact received by alleged receiver.
- **Revocation:** An instrument which takes away certain powers or privileges of an individual who had been given certain powers or privileges. Or retraction of certification or authorization [2].

1.2. Security information needs cryptography

Over the centuries, an elaborate set of protocols and mechanisms has been created deal with information security issues when the information is conveyed by physical documents. Often the objectives of information security cannot 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 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. For example, paper currency requires special inks and material 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, some wireless. 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 [2]. 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. 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 redressed; it cannot 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. There are many aspects to security and many applications, ranging from secure commerce and payments to private communications and protecting passwords. One essential aspect for secure communications is that of cryptography [29].

1.3. Models for evaluating security

The security of cryptographic primitives and protocols can be evaluated under several different models. The most practical security metrics are computational, and provable, although the latter is often dangerous. The confidence level in the amount of security provided by a primitive or protocol based on computational or ad hoc security increases with time and investigation of the scheme. However, time is not enough if few people have given the method careful analysis.

- **Unconditional security**

  The most stringent measure is an information theoretic measure whether or not a system has unconditional security. An adversary is assumed to have unlimited computational resources, and the question is whether or not there is enough information available to defeat the system. Unconditional security for encryption systems is called perfect secrecy. For perfect secrecy, the uncertainty in the plaintext, after observing the ciphertext, must be equal to the a priori uncertainty about the plaintext observation of the ciphertext provides no information whatsoever to an adversary. A necessary condition for a symmetric key encryption scheme to be unconditionally secure is that the key be at least as long as the message. The one-time pad is an example of an unconditionally secure encryption algorithm. In general, encryption schemes do not offer perfect secrecy, and each ciphertext character observed decreases the theoretical uncertainty in the plaintext and the encryption key. Public key encryption schemes cannot be unconditionally secure since, given a ciphertext c, the plaintext can in principle be recovered by encrypting all possible plaintexts until c is obtained.

- **Complexity-theoretic security**

  An appropriate model of computation is defined and adversaries are modeled as having polynomial computational power. An objective is to design a cryptographic method based on the weakest assumptions possible anticipating a powerful adversary. Asymptotic analysis and usually also worst case analysis is used and so care must be exercised to determine when proofs have practical significance. In contrast, polynomial attacks which are feasible under the model might, in practice, still be computationally infeasible. Security analysis of this type, although not of practical value in all cases, may nonetheless pave the way to a better overall understanding of security. Complexity theoretic analysis is invaluable for formulating fundamental principles and confirming intuition. This is like many other sciences, whose practical techniques are discovered early in the development, well before a theoretical basis and understanding is attained.

- **Provable security**

  A cryptographic method is said to be provably secure if the difficulty of defeating it can be shown to be essentially as difficult as solving a well known and supposedly difficult (typically number-theoretic) problem, such as integer factorization or the computation of discrete logarithms. Thus, "provable" here means provable subject to assumptions. This approach is considered by some to be as good a practical analysis technique as exists. Provable security may be considered part of a special sub-class of the larger class of computational security considered next.

- **Computational security**

  This measure the amount of computational effort required, by the best currently known methods, to defeat a system; it must be assumed here that the system has been well studied to determine which attacks are relevant. A proposed technique is said to be computationally secure if the perceived level of computation required to defeat it exceeds, by a comfortable margin, the computational resources of the hypothesized [3].

1.4. Cryptography information

Cryptography is the study of mathematical techniques related to aspects of information security such as confidentiality, data integrity, entity authentication, and data origin authentication. Or cryptography is the science of writing in secret code and is an ancient art; the first documented use of cryptography in writing dates back to circa 1900 B.C. when an Egyptian scribe used non-standard hieroglyphs in an inscription. Some experts argue that cryptography appeared spontaneously sometime after writing was invented, with applications ranging from diplomatic missives to war-time battle plans. It is no surprise, then, that new forms of cryptography came soon after the widespread development of computer communications. In data and telecommunications, cryptography is necessary when communicating over any untrusted medium, which includes just about any network, particularly the Internet [29].

1.4.1 Basic definitions of cryptographic system

- **Plaintext**: is an original intelligible message or data that fed into the algorithm as input.
- **Enciphering algorithm:** the encryption algorithm performs various substitution and transformation on plaintext.
- **Key:** is sequence of symbols that controls the operation of a cryptographic transformation. In practice a key is normally a string of bits used by a cryptographic algorithm to transform plain text into cipher text or vice versa. The key should be only part of the algorithm that it is necessary to keep secret.
- **Ciphertext:** this is scramble message produce as output. It depends on the plaintext and secret key. For given message, tow different keys will produce two different ciphertext
- **Deciphering algorithm**: This is essentially the encryption algorithm run is reverse. It takes the ciphertext and the secret key and produces the original plaintext.
- **Cryptosystem:** is a communication system designed for transmitting information securely [25].
- **Cryptology:** is all-inclusive term for study of secrecy systems in communication over nonsecure channels and related problems. In anther words cryptology is computation between cryptography and cryptanalysis [30].

1.4.2 Independent characteristic of Cryptographic system

1. **The type of operation used for transforming plaintext to ciphertext**

   All encryption algorithms are based on tow general principles: Substitution and Transposition.

2. **The number of key used**

   If both sender and receiver use the same key, the system is referred to as symmetric, single-key, secret-key, or conventional encryption. If the sender and receiver each use a different key, the system is referred as asymmetric or public-key encryption.

3. **The way in which the plaintext is processed.**

   There are tow processes

   - A block cipher processes: which the input one block of element at the time, producing an output block for each input block.
   - A stream cipher processes the input elements continuously, producing output one element at the time, as it goes along [3].

1.4.3. Operations used by Cryptographic System

Although the methods of encryption/decryption have changed dramatically since the advent of computers, there are still only two basic operations that can be carried out on a piece of plaintext - substitution and transposition. The only real difference is that whereas before these were carried out with the alphabet, nowadays they are carried out on binary bits. The operations are:

- **Substitution:** Substitution operations replace bits in the plaintext with other bits decided upon by the algorithm, to produce ciphertext. This substitution then just has to be reversed to produce plaintext from ciphertext.
- **Transposition:** Transposition (or permutation) does not alter any of the bits in plaintext, but instead move their positions around within it. If the resultant ciphertext is then put through more transpositions, the end result is increasingly secure.
- **XOR:** is an exclusive-or operation. It is a Boolean operator such that if 1 of two bits is true, then so is the result, but if both are true or both are false then the result is false.

1.4.4. Cryptographic goals

Of all the information security objectives listed in section 1.2.1, the following four form a framework upon which the others will be derived: privacy or confidentiality, data integrity, authentication and, non-repudiation.

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.

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.

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 two major classes: entity authentication and data origin authentication. Data origin authentication implicitly provides data integrity (for if a message is modified, the source has changed).

Non-repudiation: is a service, which prevents an entity from denying previous commitments or actions. When disputes arise due to an entity denying that 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 was granted. A procedure involving a trusted third party is needed to resolve the dispute. A fundamental goal of cryptography is to adequately address these four areas in both theory and practice. Cryptography is about the prevention and detection of cheating and other malicious activities.

1.5. Cryptanalysis and attacker:

There are two general approaches to attacking cryptosystem scheme Cryptanalysis and Brut-force attack.

1.5.1. Cryptanalysis

Cryptanalysis strictly, includes methods and techniques of recovering information from encrypted material without knowledge of the key, or Cryptanalysis is the process of breaking someone else's cryptographic writing. This sometimes involves some kind of statistical analysis of a passage of encrypted text. Someone who performs cryptanalysis is

called a cryptanalyst. In reality, cryptanalysis includes any method which can do so, and the most successful attacks on well implemented modern crypto systems are almost certainly not strictly cryptanalytic ones. Cryptanalytic attacks rely on the nature of the algorithm plus perhaps some knowledge of general characteristics of the plain text or even some sample plaintext-ciphertext pairs. This type of attack exploits the characteristics of the algorithm to attempt to deduce specific plaintext or to deduce the key being used. If the attack succeeds in deducing the key, the effect is catastrophic: All future and past message encrypted with that key are composed. There are many cryptanalytic techniques. Some of the more important ones for a system implementer are described below.

- **Ciphertext-only attack:** This is the situation where the attacker does not know anything about the contents of the message, and must work from ciphertext only. In practice it is quite often possible to make guesses about the plaintext, as many types of messages have fixed format headers. Even ordinary letters and documents begin in a very predictable way. For example, many classical attacks use frequency analysis of the ciphertext, however, this does not work well against modern ciphers. Modern cryptosystems are not weak against ciphertext-only attacks, although sometimes they are considered with the added assumption that the message contains some statistical bias.

- **Known-plaintext attack:** The attacker knows or can guess the plaintext for some parts of the ciphertext. The task is to decrypt the rest of the ciphertext blocks using this information. This may be done by determining the key used to encrypt the data, or via some shortcut. One of the best known modern known-plaintext attacks is linear cryptanalysis against block ciphers.

- **Chosen-plaintext attack:** The attacker is able to have any text he likes encrypted with the unknown key. The task is to determine the key used for encryption. A good example of this attack is the differential cryptanalysis which can be applied against block ciphers (and in some cases also against hash functions).

- **Man-in-the-middle attack:** This attack is relevant for cryptographic communication and key exchange protocols. The idea is that when two parties, A and B, are exchanging keys for secure communication, an adversary positions

himself between A and B on the communication line. The adversary then intercepts the signals that A and B send to each other, and performs a key exchange with A and B separately. A and B will end up using a different key, each of which is known to the adversary. The adversary can then decrypt any communication from A with the key he shares with A, and then resends the communication to B by encrypting it again with the key he shares with B. Both A and B will think that they are communicating securely, but in fact the adversary is hearing everything. The usual way to prevent the man-in-the-middle attack is to use a public key cryptosystem capable of providing digital signatures. For set up, the parties must know each others public keys in advance. After the shared secret has been generated, the parties send digital signatures of it to each other. The man-in-the-middle can attempt to forge these signatures, but fails because he cannot fake the signatures.

- **Correlation** between the secret key and the output of the cryptosystem is the main source of information to the cryptanalyst. In the easiest case, the information about the secret key is directly leaked by the cryptosystem. More complicated cases require studying the correlation (basically, any relation that would not be expected on the basis of chance alone) between the observed (or measured) information about the cryptosystem and the guessed key information.

- **Attack against or using the underlying hardware:** in the last few years as more and smaller mobile crypto devices have come into widespread use, a new category of attacks has become relevant which aim directly at the hardware implementation of the cryptosystem. The attacks use the data from very fine measurements of the crypto device doing, say, encryption and compute key information from these measurements. The basic ideas are then closely related to those in other correlation attacks. For instance, the attacker guesses some key bits and attempts to verify the correctness of the guess by studying correlation against her measurements[4].

1.5.2. Brut-force attack

What is a brute force attack?

The attacker tries every possible key on piece of ciphertext until an intelligible translation into plaintext is obtained. On average, half of all possible keys must be tried to achieve success [3]. Millions of keys are used in simultaneous attempts to decrypt the ciphertext, assuming the decryption method is known and the result in each case is tested to ascertain whether it is something intelligible. Brute force attacks against specific cryptosystems can be compared since the average time required by a brute force attack is half the number of possible keys multiplied by the time required to test each key. It is true that if the size of the key space associated with a cryptosystem is small (e.g. 2^{16} = 65,536) then the cryptosystem is vulnerable to a brute force attack. But if a cryptosystem has a very large key space (e.g. about (10^{100}) if a 60-character key is permitted) then a brute force attack is not feasible and so any weakness in the system, if it exists, must be sought elsewhere [37].

What can be done to prevent brute force attack?

The simplest thing to prevent brute force attacks from succeeding is to choose truly random keys that are not based on words in any language and are longer than 8 letters. The longer the key, the longer it would take a brute force tool to find, by not using a word (or a pattern based on a word) you eliminate the chance of a dictionary attack working; by using a truly random sequence of letters, numbers, and characters you ensure that only an exhaustive search will find your key, giving you the advantage of time. The second most important thing is to change your keys frequently. If someone does guess your key you can limit how long they have access to your account by frequently changing your key [38].

1.6. Summary

The data is very important in our delay life and it must be secure from attack or from lost. To achieve security to date we need cryptography. There are many benefits of security like Privacy or confidentiality, Data integrity, Entity authentication or identification, Message authentication, and Signature. There are many goals of cryptograph like Confidentiality, Data integrity, Authentication, and Non-repudiation Privacy or confidentiality, Data integrity, Entity authentication or identification, Message authentication, and Signature.

2. CRYPTOGRAPHY ALGORITHMS

2.1. Overview

A cryptography algorithm is the mathematical function used for encryption and decryption.

There are two basic kinds of encryption algorithms in use today

- Private key cryptography, which uses the same key to encrypt and decrypt the message. This type is also known as symmetric key cryptography.

- Public key cryptography, which uses a public key to encrypt the message and a private key to decrypt it. The name public key comes from the fact that you can make the encryption key public without compromising the secrecy of the message or the decryption key. Public key systems are also known as asymmetric key cryptography [26].

2.2. Restricted Algorithm.

If the security of the message being sent relies on the algorithm itself remaining secret, then that algorithm is known as a restricted algorithm.

The problem or drawback of restricted algorithm

The algorithm obviously has to be restricted to only those people that you want to be able to decode your message. Therefore a new algorithm must be invented for every discrete group of users or in other meaning restricted algorithms allows no quality control or standardization. Every group of users must have their own unique algorithm. They have to write their own algorithms and implementations. A large or changing group of users cannot use them, as every time one user leaves the group, everyone must change algorithm, so that people outside the group won't know the top secret (algorithm itself). If someone accidentally finds out the secret how the algorithm works, the group must change their algorithm. If the algorithm is compromised in any way, a new algorithm must be implemented. Modern cryptography or key based algorithm solves the problems that are presented by Restricted Algorithm [23].

2.3. Key Based Algorithms.

There are two categories of cryptographic key based algorithms: conventional and public key.

2.3.1 Symmetric cryptography algorithms

Symmetric cryptography, also known as Conventional cryptography, requires that the sender and receiver have shared a key, which is a secret piece of information that is used to encrypt or decrypt a message. If this key is secret, then nobody other than the sender or receiver can read the message [26]. Using Figures 2.1 below explains how symmetric key work.

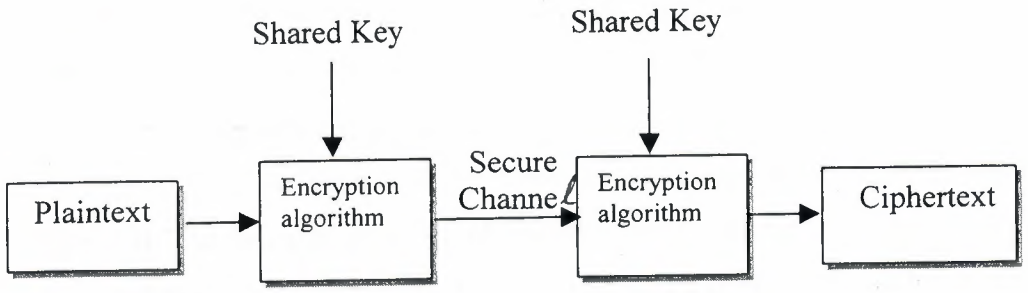

Figure 2.1 Model of Symmetric cryptosystem

Suppose a message in plaintext is X, K is shared key used for the same for sender and receiver, mean for encryption and decryption, and Y is ciphertext so we can get cipher-text through this formula,

$$Y=E_K(X).$$

This formula indicates that Y (cipher text) is obtained by using encryption algorithm E as function of plaintext X. In the receiver receive the ciphertext (Y) and the shared key and used decryption algorithm (D), which is the inverse of E to obtain ciphertext,

$$X=D_K(Y).$$

Opponent, observing Y but not having access to k or X, may attempt to recover X or K or both of them [1].

2.3.1.1. Symmetric algorithms operations

- **S-boxes:** Lookup tables that map n bits to m bits (where n and m often are equal). There are several ways of constructing good S-boxes for ciphers, as well as several ways of measuring them. Some designers use a rigorous mathematical approach to create S-boxes, which can be proven to be resistant against some particular attacks. Other designers use heuristic approaches, which may lead to S-boxes that are more difficult to handle in mathematical proofs [4].

- **Feistel networks:** A Feistel network is a general device of constructing block ciphers from simple functions. The original idea was used in the block cipher Lucifer, invented by Horst Feistel. Several variations have been devised from the original version. Put simply, the standard Feistel network takes a function from n bits to n bits and produces an invertible function from 2n bits to 2n bits. The basic function upon which the structure is based is often called the round function. The essential property of Feistel networks that makes them so useful in cipher design. The security of the Feistel structure is not obvious, but analysis of DES has shown that it is a good way to construct ciphers.

- **The operation** of taking the user key and expanding it. Key expanding in Feistel rounds is called key scheduling. This is often a non-linear operation.

- **Expansion, Permutation:** These are common tools in mixing bits in a round function. They are linear operations.

- **Bitslice operations:** is bitwise logic operations XOR, AND, OR, NOT and bit permutations. The idea of bitslice implementations of block ciphers is due to Eli Biham. It is common practice in vector machines to achieve parallel operation.

- **Pseudo-Hadamard Transformation:** A pseudo-Hadamard transform (PHT) is a simple mixing operation that runs quickly in software. Given two inputs, a and b, the 32-bit PHT is defined as:

$$a' = a + b \bmod 2^{32},$$

$$b' = a + 2b \bmod 2^{32}$$ [8].

- **MDS Matrices:** A maximum distance separable (MDS) code over a field is a linear mapping from field elements to b field elements, producing a composite vector of a + b elements, with the property that the minimum number of non-zero elements in any non-zero vector is at least b + 1. Between any two distinct vectors produced by the MDS mapping is at least b + 1. It can easily be shown that no mapping can have a larger minimum distance between two distinct vectors, hence the term maximum distance separable. MDS mappings can be represented by an MDS matrix consisting of a x b elements. Reed-Solomon (RS) error-correcting codes are known to be MDS. A necessary and sufficient condition for an a x b matrix to be MDS is that all possible square sub matrices, obtained by discarding rows or columns, are non-singular [8].

- **Whitening:** Whitening, the technique of xoring key material before the first round and after the last round. Whitening substantially increases the difficulty of key search attacks against the remainder of the cipher.

2.3.1.2. Classical algorithms

Caesar cipher: one of the earliest cryptosystem is often attributed to Julius Caesar. Cease cipher depend on shift the letter by value. For example suppose if the shift value equal to 3.

Plaintext	T	H	I	S	I	S	M	Y	T	H	E	S	I	S
Shift value	3													
Ciphertext	W	K	L	V	L	V	P	B	W	K	H	V	L	V

Means that cipher text is obtain by

$$C=E(P)=(P + n) \bmod (26).$$

Where (n) shift value, and its rang between [1-24]. Plaintext can obtain by

$$P=D(C)=(C - n) \bmod (26)$$ [3].

Hill Cipher: Developed by Lester Hill in 1992. The core of Hill cipher is matrix manipulations. Both encryption and decryption are nothing more than matrix multiplications, the encryption algorithm takes m successive plaintext letters and

substitutes for them m ciphertext letters. The substitutes is determine by m linear equations in which each character is assigned a numerical value (a=0, b=1, ...,z = 25). For m = 3, the system can be described as follows:

$$c_1 = (k_{11}P_1 + k_{11}P_1 + k_{11}P_1) \bmod 26$$
$$c_2 = (k_{11}P_1 + k_{11}P_1 + k_{11}P_1) \bmod 26$$
$$c_3 = (k_{11}P_1 + k_{11}P_1 + k_{11}P_1) \bmod 26$$

This can be expressed in term of column vectors

$$\begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} k_{11}\ k_{12}\ k_{13} \\ k_{21}\ k_{22}\ k_{23} \\ k_{31}\ k_{32}\ k_{33} \end{pmatrix} \begin{pmatrix} P_1 \\ P_2 \\ p_3 \end{pmatrix} \bmod \quad 26$$

Or can expressed as C = KP mod 26 [1, 36].

Vigenère Cipher: the Vigenère cipher is a polyalphabetic cipher based on using successively shifted alphabets, a different shifted alphabet for each of the 26 English letters. The Vigenère tableau in table 2.1 can be used directly. For each letter of the message use the letter of the keyword to determine a row and go across the row to the column headed by the corresponding letter of the message. We illustrate this row-column look-up.

Example To Vigenère Cipher:

Keyword (Row)	RELAT	IONSR	ELATI	ONSRE	LATIO	NSREL
Plaintext (column)	TOBEO	RNOTT	OBETH	ATIST	HEQUE	STION
Cipher Text	KSMEH	ZBBLK	SMEMP	OGAJX	SEJCS	FLZSY

Now there are many symmetric algorithms can be historical divided into a Classical algorithm and recent algorithm.

	a	b	c	D	e	f	j	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
a	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
b	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
c	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
d	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
e	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
f	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
g	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
h	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
i	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
j	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
k	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
l	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
m	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
n	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
o	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
p	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
r	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
s	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
t	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
u	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
v	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
w	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
x	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Table 2.1 The Vigenère Tableau [1].

Playfair cipher:

In 1854, Sir Charles Wheatstone invented the cipher known as "Playfair" named for his friend Lyon Playfair, first Baron Playfair of St. Andrews, who popularized and promoted the cipher. Its simplicity and its cryptographic strength compared to simple substitution and Vigenère (a polyalphabetic substitution cipher) made it an immediate success as a field cipher, used by the British in the Boer War and the First World War, and by several armed forces as an emergency back-up cipher in the Second World War. When Lt. John F. Kennedy's PT-109 was sunk by a Japanese cruiser in the Solomon Islands, for instance, he made it to shore on Japanese-controlled Plum Pudding Island and was able to send an emergency message in Playfair from an Allied coast-watcher's hut to arrange the rescue of the survivors from his crew [34].

2.3.1.3 Recent Algorithms

DES: The US National Bureau of Standards (NSB) published the Data Encryption Standard in 1975. Created by IBM, DES came about due to a public request by the NSB requesting proposals for a standard cryptographic algorithm that satisfied the following criteria:

- The security depends on keys, not the secrecy of the algorithm
- The security is capable of being evaluated
- The algorithm is completely specified and easy to understand
- It is efficient to use and adaptable
- Must be available to all users
- Must be exportable
- Provides a high level of security

DES has now been in world-wide use for over 20 years, DES is symmetric, block-cipher algorithm with a key length of 64 bits [1].

IDEA: Was created in its first form by Xuejia Lai and James Massey in 1990, this was called the Proposed Encryption Standard (PES). In 1991, Lai and Massey strengthened the algorithm against differential cryptanalysis and called the result Improved PES (IPES). The name of IPES was changed to International Data Encryption Algorithm

(IDEA) in 1992. IDEA is a symmetric, block-cipher algorithm with a key length of 128 bits, a block size of 64 bits.

Five candidate algorithms for Advanced Encryption Standard

On January 2, 1997, the National Institute of Standard and Technology (NIST) invited proposals for new algorithm for Advanced Encryption Standard to replaced the old Data Encryption standard (DES). Among the 15 preliminary candidates, MARS, RC6, Renal, Serpent, and Two fish were announced as the finalist candidates on August 9, 1999 for further evaluation. After studying all information and public comments on these finalist candidates, NIST announced in October 2000 that Rijndeal was the selected as the AES algorithm [5].

1. **Rijndael:** The cipher Rijndael is one of the five finalists of the Advanced Encryption Standard. Joan Daemen and Vincent Rijmen have designed the algorithm. It is a block cipher. The length of the block and the length of the key can be specified to be 128, 192, 256 bits [10].

2. **Serpent:** Is a 128-bit block cipher has been designed by Ross Anderson, Eli Biham and Lars Knudsen. Serpent is considering a candidate for the Advanced Encryption Standard algorithm [12].

3. **Twofish:** Developed by Counterpane Systems, Two fish is a 128-bit block cipher that accepts a variable-length key up to 256 bits. The cipher is a 16-round Feistel network [8].

4. **RC6:** developed by Ron Rivest, is a new block cipher submitted to NIST for consideration as the new Advanced Encryption Standard (AES). RC6 is more accurately specified as RC6-w/r/b where the word size is w bits, encryption consists of a nonnegative number of rounds r, and b denotes the length of the encryption key in bytes [9].

5. **MARS:** developed by IBM, a shared-key (symmetric) block cipher supporting 128-bit blocks and variable key size. MARS is designed to take advantage of the

powerful operations supported in today's computers, resulting in a much improved security/performance tradeoff over existing ciphers [11].

2.3.2. Asymmetric cryptography algorithms

Asymmetric cryptography, also know as Public key cryptography were invented in the late 1970's, with some help from the development of complexity theory.

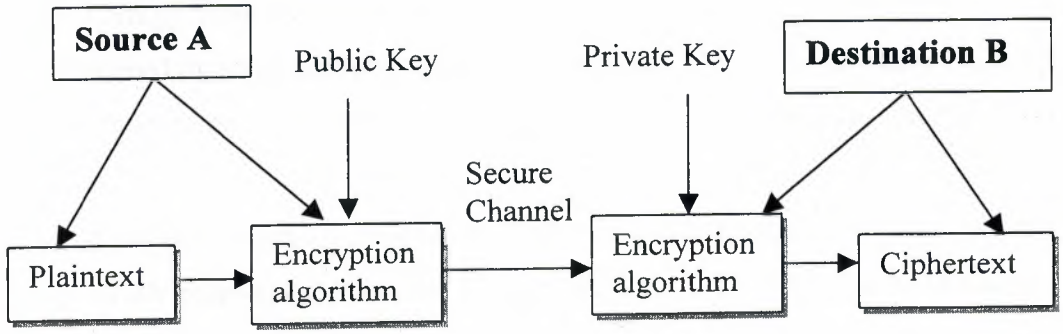

Figure 2.2 Model of Asymmetric Cryptosystem

Around that time, Asymmetric cryptography solves the key exchange problem by defining an algorithm, which uses two keys, one of them can be used to encrypt, usually (public key), a message, and another key, usually private key, used to decrypt a massage. Using Figures 2.2 explain how asymmetric key work. Suppose the source A that produces a message or plaintext is X, the message is intended for destination B, B generates a related pair of keys: a public key, K_{Ub}, and private key KRb. KRb is known to only to B, whereas K_{ub} is public available and accessible by A. with message X, E encryption algorithm and encryption key K_{Ub} as input to get the ciphertext Y using this formula, $Y=E_{Kub}(X)$, The intended receiver use its private key to be able to invert transformation, $X=D_{KRb}(Y)$.

Opponent, observing Y and having access to K_{Ub} but not having access to K_{Rb} or X, may attempt to recover X or KRb or both of them. The basic ingredient in any public key cryptosystem is a difficult computational problem. The security of the cryptosystem is based on the fact that the private key can be computed from the public key only by solving this difficult problem.

2.3.2.1. Asymmetric algorithms operations

- **Algorithm:** algorithm is an explicit description how a particular computation should be performed. The efficiency of an algorithm can be measured as the number of elementary steps it takes to solve the problem. So if we claim that the algorithm takes time O (n) then we mean that it takes n elementary steps, but we do not specify how long one step takes.

- **Computational complexity:** A problem is polynomial time or in P if it can be solved by an algorithm, which takes less than O (nt) steps, where t is some finite number and the variable n measures the size of the problem instance.

- **Primes:** A prime number is a number that has no divisors except for itself and on the number (1), thus the integers 2, 3, 5, 7, 11, 13, 17, 19, 23 and so on are primes.

- **Factoring:** Every integer can be represented uniquely as a product of prime numbers. For example, 10 = 2 * 5 and it is unique. The art of factorization is almost as old as mathematics itself. However, the study of fast algorithms for factoring is only a few decades old. One possible algorithm for factoring an integer is to divide the input by all small prime numbers iteratively until the remaining number is prime.

- **Discrete logarithms:** Another important class of problems is the problem of finding n given only some y such that y = gn. The problem is easy for integers, but when we are working in a slightly different setting it becomes very hard.

- **Knapsacks:** given a small set of integers, the knapsack problem consists of determining a subset of these integers such that their sum is equal to a given integer. For example, given (2, 3, 5, 7) and 10, we can find the solution 2 + 3 + 5 = 10, and thus solved the knapsack problem, by brute force search [4].

2.3.2.2. Common Public key Algorithms

RSA: The Rivest, Shamire and Adelmen (RSA) design for a public key cipher in 1978. RSA is based on the use of product of two very large number prime, and relying on the fact that the determination of the prime factors of such large numbers is so computationally difficult as to be effectively impossible to compute [7].

Diffie-Hellman: Is a commonly used protocol for key exchange. In many cryptographically protocols two parties wish to begin communicating. However, assume they do not initially possess any common secret and thus cannot use secret key cryptosystems. The key exchange by Diffie-Hellman protocol remedies this situation by allowing the construction of a common secret key over an insecure communication channel. It is based on a problem related to discrete logarithms, namely the Diffie-Hellman problem. This problem is considered hard, and it is in some instances as hard as the discrete logarithm problem. The Diffie-Hellman protocol is generally considered to be secure when an appropriate mathematical group is used.

ElGamal: ElGamal is public key cipher. This is a straightforward extension of Diffie-Hellman's original idea on shared secret generation. Essentially, it generates a shared secret and uses it as a one-time pad to encrypt one block of data.

Elliptic Curve: Is just another way of implementing discrete logarithm methods. An elliptic curve is basically a set of points that satisfy the equation $y^2 = x^3 + ax + b$ when considered in finite field of characteristic p (where p must be larger than 3). A slightly different equation is needed for the cases of small characteristic, $p = 2$ and $p = 3$. The points on elliptic curves can be added together and they form a structure called a group. This is just a way of saying that you can do arithmetic with them as you can do with integers when using just addition and subtraction.

2.4. One-Time Pads.

2.4.1. Definition of One-Time Pads

The one-time pad was invented by Major Joseph Mauborgne and Gilbert Bernam in 1917, and is an unconditionally secure symmetric algorithm [5]. The theory behind a one-time pad is simple. The pad is a non-repeating random string of letters. Each letter

on the pad is used once only to encrypt one corresponding plaintext character means that the length of bad same length of message. After use the pad must never be re-used. As long as the pad remains secure, so is the message. This is because a random key added to a non-random message produces completely random ciphertext and ciphertext bears no statistical relationship to plaintext, and there is absolutely no amount of analysis or computation that can alter that. If both pads are destroyed then the original message will never be recovered [23].

2.4.2. How One-time Pad works

To use a one-time pad you need tow copies of the pad, there are two pads issued to each user. One for encipher and one for decipher, typically the pads are set up in blocks of five letter random groups. The key text may not be reused and to use the OTP, a method is needed for correlating a letter of plaintext with the next letter of the key text (from the pad), to produce a letter of enciphered text. The method used is called a "Vigenere's Tableau", as discus before. The table has the alphabet in the left-most column, and also across the top. For each row, there is a shifted-reverse alphabet. To encipher the first letter in a message, go to the row corresponding to the plain-text letter, then go to the column indicated by the first letter on your OTP. The letter at the row-column intersection is the enciphered letter. Note that the Vigenere's table itself does not contain any secret information, it simply provides the mechanism for combining plain and key text into enciphered text. For example, suppose that the message is "Dead drop Alpha three AM tonight" [1, 2, 23].

Plain Text	DEADD	ROPAL	PHATH	REEAM	TONIG	HTXXX
Text from OTP	BNJEX	KQPBC	LZCXV	PKTUY	QFHNG	QWERT
Encipher Text	VIQSZ	YVVYM	ZTXJX	TLCFP	QGFEN	CKYLJ

2.4.3. Drawback of one-time pads

It is extremely hard to generate truly random numbers, and a pad that has even a couple of non-random properties is theoretically breakable. Secondly, because the pad can never be reused no matter how large it is, the length of the pad must be the same as the length of the message - fine for text, but virtually impossible for video.

2.5. Summary

Public key (symmetric algorithm) and private key(asymmetric algorithm) are the most important cryptography algorithms, Symmetric algorithms can be divided into stream ciphers and block cipher. The most popular symmetric-key system is the Data Encryption Standard (DES), and now days it replaced Advanced Encryption Standard (AES). Public key algorithms use a different key for encryption. The most popular asymmetric-key system is RSA (Rivest-Shamir-Adelman) developed by Ron Rivest. A symmetric algorithm is faster than symmetric algorithm [29].

3. FIVE CANDIDATES ALGORITHMS FOR AES

3.1. Overview

The AES (Advanced Encryption Standard) is symmetric-key block cryptographic algorithm that was approved by the US National Institute of Science and Technology (NIST) as a replacement for the Data Encryption Standard (DES), which had been approved for the encryption of financial information since the late 1970's.In the late 1990's DES was felt by many experts to be vulnerable to brute force attack by specialized computing machines. NIST held a public competition for replacement DES, and a number of algorithms were proposed. Among the 15 preliminary candidates, MARS, RC6, Rijndael, Serpent, and Twofish were announced as the finalist candidates on August 9, 1999 for further evaluation. These five algorithms are well balanced looking to the various evaluation criteria of expected and proposed security, implementation and performance characteristics. Selecting all 5 as AES-FIPS (Federal Information Processing Standard) is inappropriate with initial goal of NIST. After studying all information and public comments on these finalist candidates, NIST announced in October 2000 that Rijndael was the selected as the AES algorithm [5].

3.2. Rijndael algorithm

3.2.1. Introduction to Rijndael

The Rijndael proposal for AES defined a cipher in which the block length and the key length can be independently specified to be 128, 192, 256 bits. The AES specification uses the same three key size alternatives but limits the block length to 128 bits. Table 3.1 shows that the numbers of Rijndael parameter depends on the size of key [1].

Table 3.1 AES Parameters.

Key size (words/bytes/bits)	4/16/128	6/24/192	8/32/256
Plaintext block size (words/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	52/128	60/240

To explains this table. When the key is 128 bits is equivalents to 16 bytes or equivalents to 4 words because the word contains four bytes. Other will explain later.

State array

Rijndael operates on a two dimensional array of bytes called the state that contains 4 rows and *Nk* columns, where *Nk* is the length of key length divided by 32. This state array, denoted by the symbol *S*, each individual byte has two indexes: its row Number r, in the range $0 < r < 4$, and its column number *c*, in the range $0 < c < Nk$, hence allowing it to be referred to either as $S_{r,c}$ or S[r, *c*]. For AES the range for *c* depend on the key means c maybe 4,6,8 when the length of key is 128, 192, 256 bits, respectively. At the start or end of an encryption or decryption operation the bytes of the cipher input or output are copied to or from this state array in the order shown in Figure 3.1 show when key is 128 so the number of row is 4 and each` row contain 4 bytes.

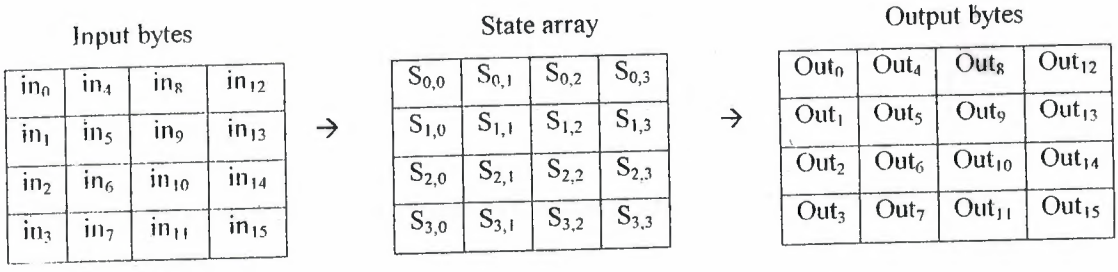

Figure 3.1 mapping of input bytes, State array and output bytes [5].

Hence at the start of encryption or decryption the input array in is copied to the state array according to the scheme:

$$s[r, c] = in[r + 4c] \quad \text{for } 0 < r < 4 \text{ and } 0 < c < Nk.$$

And when the cipher is complete the state is copied to the output array, out, according to the scheme:

$$out[r + 4c] = s[r, c] \quad \text{for } 0 < r < 4 \text{ and } 0 < c < Nk.$$

The AES algorithm is alternative algorithm. The total number of round (iteration), Nr, is 10 when Nk=4, Nr=12 when Nk=6, and Nr=14 when Nk=8.

3.2.2. Construction of the S-Box

First computing the multiplicative inverse of each element in GF(2^8) with irreducible polynomial $m(x)=x^8 + x^4 + x^3 + x + 1$. Second the affine transformation over GF(2) defined by:

$$b`_i = b_i \oplus b_{(i+4)\bmod 8} \oplus b_{(i+5)\bmod 8} \oplus b_{(i+6)\bmod 8} \oplus b_{(i+7)\bmod 8} + C_i \text{ [10].}$$

For $0 < i < 8$ where b_i is bit i of the byte and C_i is bit i of a byte C with the value {63} or {01100011}. Here and elsewhere a prime on a variable on the left of an equation indicates that its value is to be updated with the value on the right. In matrix form the latter component of the S-box transformation can be expressed as: [3].

$$\begin{pmatrix} b`_0 \\ b`_1 \\ b`_2 \\ b`_3 \\ b`_4 \\ b`_5 \\ b`_6 \\ b`_7 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \\ b_8 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

The final result of this two-stage transformation is given in the following S-box

Table 3.2 S-Box For Rijndael

		X															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Y	0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
	1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
	2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
	3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
	4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
	5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
	6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7E	50	3C	9F	A8
	7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
	8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	46	5D	19	73
	9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
	A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
	B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
	C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
	D	70	38	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
	E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
	F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

3.2.3. Key Expansion

In AES algorithm, Key expansion generates a total of (Nb * (Nr+1)) words, where Nb is number of bytes in the rows. The Key, K, is used as the initial set of Nk words, and the reset of the words are generated from the key iteratively. The output of key expansion is an array of 4-byte words denoted by w_i, where $0 \leq i < Nb\ (Nr+1)$. Each RoundKey is a concatenation of 4 words from the output of key expansion, round(i)=(w_{4i}, w_{4i+1}, w_{i+2}, w_{i+3}). Figure 3.2 show pseudo code foe key expansion [10].

```
KeyExpansion (byte key (4*Nk), word w (Nb*(Nr+1)), NK)
begin
word temp
i=0
while (i<NK)
    w(i)= word [ key[4*i+3], key[4*i+2], key[4*i+1], key[4*i] ]
    i = i + 1
end while
i = Nk
while (i < Nb * (Nr + 1))
    word temp = k[i - 1]
    if (i mod Nk = 0)
        temp = SubWord(RotWord(temp)) xor Rcon[i / Nk]
    else if ((Nk > 6) and (i mod Nk = 4))
        temp = SubWord(temp)
    end if
    k[i] = k[i - Nk] xor temp
    i = i + 1
end while
end
```

Figure 3.2 Pseudo code for Key Expansion

Subword in Figure 3.2 applies S-box (see Table 3.2) to each of the four bytes in a word. The function RotWord rotates each byte in a word one position to the left. For example, if the input is a word $[a_0, a_1, a_2, a_3]$, RotWord returns the word $[a_0, a_1, a_2, a_3]$. Rcon(i) is the round constant word array, whose value is [RC(i),{00}, {00}, {00}]. The values of Rc(i) are listed as follows:

RC(1)={01} RC(2)={02} RC(3)={04} RC(4)={08} RC(5)={10}
RC(6)={20} RC(7)={40} RC(8)={80} RC(9)={1b} RC(10)={36}.

3.2.4. Encryption algorithm

At the start of the cipher the cipher input is copied into the internal state using the conventions described in section 3.2.1. An initial round key is then added after that Nr round of encryption are performed. The first Nr-1 is the same, with the small different in the final round. As illustrated in Figure 3.3, each of the first Nr-1 consist of 4 transformation [5].

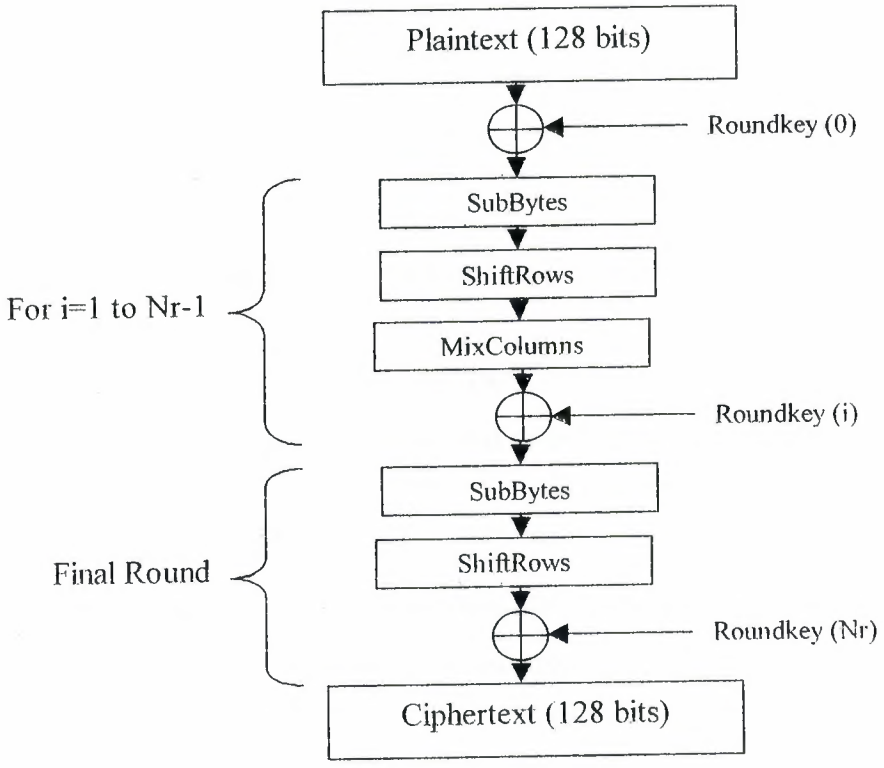

Figure 3.3 AES Encryption Structure algorithm [5].

1. SubByte Transformation

The Sub Bytes transformation is a non-linear byte substitution that acts on every byte of the state in isolation to produce a new byte value using an S-box substitution table which discus above in Table 3.2. Write a byte as 8 bit, 4 bits to determine the row and other 4 bits to determine the column, if the input is 100001011, looking in row 8 (the ninth row) and column 11 (the twelfth), the intersection between row and columns will be the rustle which is 61 or 1111101 in binary.

2. ShiftRows Transformation

The ShiftRows transformation operates individually on each of the last three rows of the state by cyclically shifting the bytes in the row to the left [2].

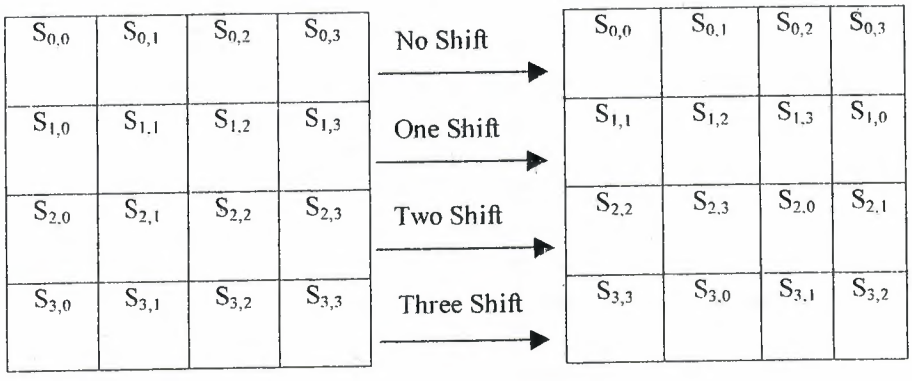

Figure 3.4 ShiftRows Transformation.

As illustrated in Figure 3.4 the first row no shift. The second, third, and fourth rows shift cyclically to the left one byte, two bytes, and three bytes, respectively.

3. MixColumns Transformation

The MixColumns transformation acts independently on every column of the state and treats each column as a four-term polynomial over $GF(2^8)$ and multiplied by a(x) modulo x^4+1 where $a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$.In matrix form this transformation can be expressed as:

$$\begin{pmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{pmatrix} = \begin{pmatrix} \{02\} & \{03\} & \{01\} & \{01\} \\ \{01\} & \{02\} & \{03\} & \{01\} \\ \{01\} & \{01\} & \{02\} & \{03\} \\ \{03\} & \{01\} & \{01\} & \{02\} \end{pmatrix} \begin{pmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{pmatrix} \quad 0 \leq c < Nk.$$

4. AddRoundKey Transformation

In AddRoundKey transformation, a round key is added to the state by bitwise Exclusive-OR (XOR) operation. Each round Key consist of Nb words generated by from key expansion describe above.

3.2.5. Decryption algorithm

The decryption structure is show below in Figure 3.5 corresponding to the transformation in the encryption, InvSubBytes, InvShifRows, InvmixColumns, and AddRoundKey are transformation used in the decryption. The round Keys are the same as those in encryption generated by key expansion, but are used in inverse order [10].

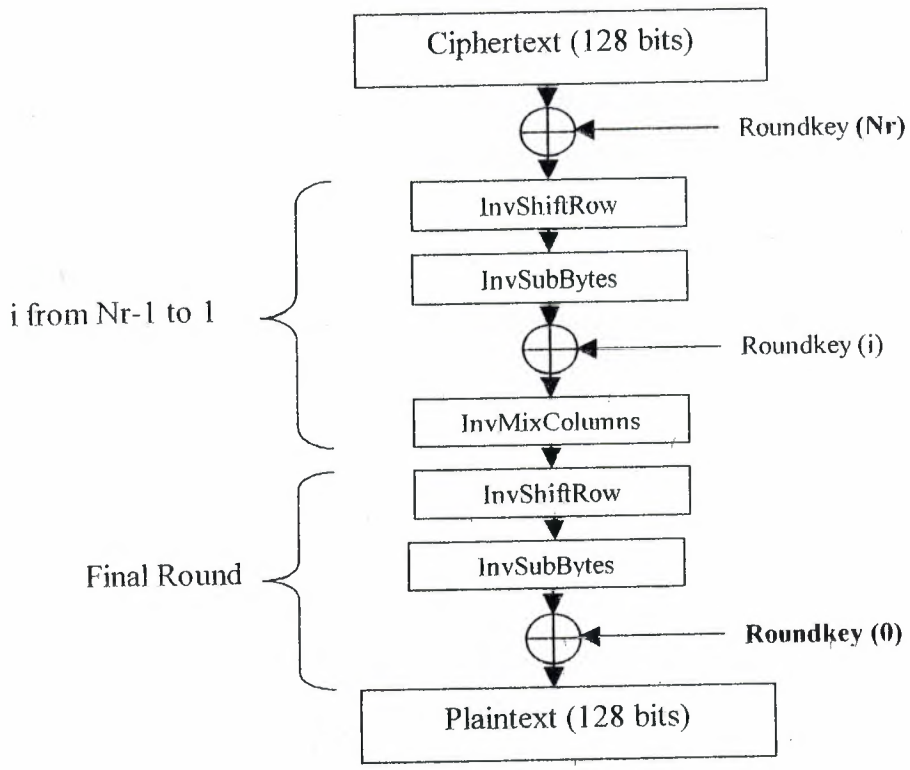

Figure 3.5 Straightforward Decryption Structure.

1. InvShiftRows Transformation

The InvShiftRows transformation operates individually on each of the last three rows of the state by cyclically shifting the bytes in the row into the right [10]

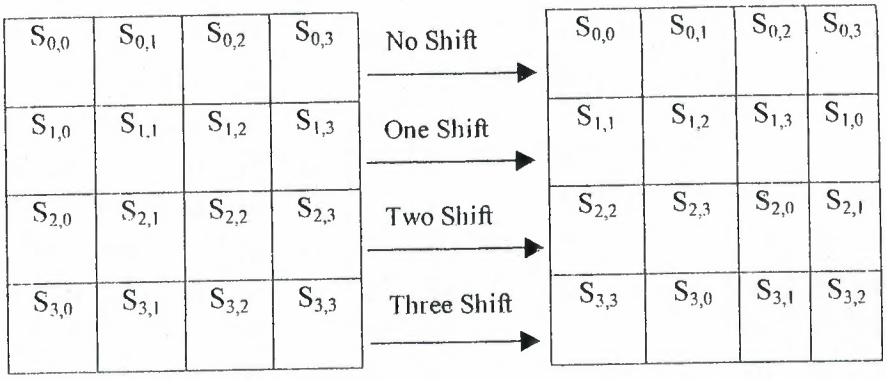

Figure 3.6 InveShiftRows Transformation.

As illustrated in Figure 3.6 the first row no shift. The second, third, and fourth rows shift cyclically to the right one byte, two bytes, and three bytes, respectively.

2. InvSubBytes Transformation

The inverse S-box table needed for the InvSubBytes transformation. The pseudo code for this transformation is as follows:

```
InvSubBytes(byte state[4,Nk], Nk)
begin
for r = 0 step 1 to 3
    for c = 0 step 1 to Nk - 1
        state[r,c] = InvSbox[state[r,c]]
    end for
end for
end
```

gives the full inverse S-box, the inverse of the affine transformation being:

$b`_i = b_{(i+2)\bmod 8i} \oplus b_{(i+5)\bmod 8} \oplus b_{(i+7)\bmod 8} \oplus d_i$ Where byte d = {05} [10].

The inverse S-Box is show below in Table 3.3.

		X															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Y	0	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
	1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
	2	54	7B	94	32	A6	C2	23	3D	EE	AC	95	0B	42	FA	C3	4E
	3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
	4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
	5	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
	6	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
	7	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
	8	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
	9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
	A	47	F1	1A	71	1D	29	20	9A	DB	C0	62	OE	AA	18	BE	1B
	B	FC	56	3E	48	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
	C	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
	D	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
	E	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
	F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

Table 3.3 Inverse S-box for Rijndael.

3. InvMixColumns Transformation

The InvMixColumns transformation acts independently on every column of the state, as similar to MixColumns transformation, and treats each column as a four-term polynomial GF(2^8) and multiplied by $a^{-1}(x)$ modulo x^4+1. Where $a^{-1}(x) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\}$. The below matrix explains the Inverse MixColumns transformation [5].

$$\begin{pmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{pmatrix} = \begin{pmatrix} \{0e\} & \{0b\} & \{0d\} & \{09\} \\ \{09\} & \{0e\} & \{0b\} & \{0d\} \\ \{0d\} & \{09\} & \{0e\} & \{0b\} \\ \{0b\} & \{0d\} & \{09\} & \{0e\} \end{pmatrix} \begin{pmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{pmatrix} \quad 0 \leq c < Nk$$

3.2.6. Security and efficiency is key to Rijndael

Rijndael is an encryption algorithm that has been designed with the state of art in the cryptographic research and is still believed very secure by many people. It has been designed to have very strong resistance against the classical approximation attacks, such as linear cryptanalysis, differential cryptanalysis. The version of AES with a key length of 128 bits is likely to be the one most commonly implemented; this length is sufficient to provide security and requires less processing time than a longer key length. Thus far there doesn't appear to be any critical weaknesses in either AES or Triple-DES, so the level of security is directly proportional to the key length.

3.3. Serpent algorithm

Serpent is a 128-bit block cipher designed by Ross Anderson, Eli Biham and Lars Knudsen as a candidate for the Advanced Encryption Standard, meaning that data is encrypted and decrypted in 128-bit chunks. The key length can vary, but for the purposes of the AES it is defined to be 128, 192, or 256 bits. This block size and variable key length is standard among all AES candidates and was one of the major design requirements specified by NIST. The Serpent algorithm uses 32 rounds, or iterations of the main algorithm. Using the largest possible key size to ensure that the user always enjoys the best possible security. Like DES, Serpent includes an initial and final permutation of no cryptographic significance, these permutations are used to optimize the data before encryption.

3.3.2. Encryption and Decryption algorithms

Serpent is a 32-round SP-network operating on four 32-bit words, thus giving a block size of 128 bits. All values used in the cipher are represented as bit streams. The indices of the bits are counted from 0 to bit 31 in one 32-bit word, 0 to bit 127 in 128-bit blocks, 0 to bit 255 in 256-bit keys, and so on. For internal computation, all values are represented in little endian, where the first word (word 0) is the least significant word, and the last word is the most significant, and where bit 0 is the least significant bit of word 0. Externally, each block write as a plain 128-bit hex number. Serpent encrypts a 128-bit plaintext P to a 128-bit ciphertext C in 32 rounds under the control of 33 128-bit subkeys $K_0, \ldots K_{32}$. The user key length is variable, but for the purposes of this submission it fix it at 128, 192 or 256 bits, short keys with less than 256 bits are mapped

to full-length keys of 256 bits by appending one "1" bit to the MSB end, followed by as many "0" bits as required to make up 256 bits. This mapping is designed to map every short key to a full-length key, with no two short keys being equivalent. The cipher itself consists of:

- An initial permutation IP
- 32 rounds, each consisting of a key mixing operation, a pass through S-boxes, and (in all but except last round) a linear transformation. In the last round, an additional key mixing operation replaces this linear transformation.
- A final permutation FP.

The initial and final permutations do not have any cryptographic significance. They are used to simplify an optimized implementation of the cipher. The initial permutation IP is applied to the plaintext P giving B'_0, which is the input to the first round. The rounds are numbered from 0 to 31, where the first round is round 0 and the last is round 31.The output of the first round is B'_1, the output of the second round is $B\hat{}_2$, the output of round i is $B\hat{}_{i+1}$, and so on, until the output of the last round (in which the linear transformation is replaced by an additional key mixing) is denoted by $B\hat{}_{32}$. The final permutation FP is now applied to give the ciphertext C. Each round function R_i ($i \in \{ 0, \ldots, 31 \}$) uses only a single replicated S-box. For example, R_0 uses S_0, 32 copies of which are applied in parallel. Thus the first copy of S_0 takes bits 0, 1, 2 and 3 of $B\hat{}_0 \oplus K\hat{}_0$ as its input and returns as output the first four bits of an intermediate vector; the next copy of S_0 inputs bits 4-7 of $B'_1 \oplus K'_1$ and returns the next four bits of the intermediate vector, and so on. The intermediate vector is then transformed using the linear transformation, giving $B\hat{}_1$. Similarly, R_1 uses 32 copies of S_1 in parallel on $B'_1 \oplus K'_1$ and transforms their output using the linear transformation, giving B'_2. The set of eight S-boxes is used four times. Thus after using S_7 in round 7, useing S0 again in round 8, then S_1 in round 9, and so on. The last round R_{31} is slightly different from the others: applying S_7 on $B'_{31} \oplus K'_{31}$, and XOR the result with K'_{32} rather than applying the linear transformation. The result B'_{32} is then permuted by FP, giving the ciphertext. Thus the 32 rounds use 8 different S-boxes each of which maps four input bits to four output bits. Each S-box is used in precisely four rounds, and in each of these it is used 32 times in parallel. The S-box design is discussed below. As with DES, the final permutation is the inverse of the initial permutation. Thus the cipher may be formally described by the following equations:

$$B'_0 = IP(P)$$

$$B'_{i+1} = R_i(B'_i)$$

$$C = FP(B'_{32})$$

Where

$$R_i(X) = L(S'_i(X \oplus K'_i)) \qquad i = 0, \ldots, 30$$

$$R_i(X) = S'_i(X \oplus K'_i) \oplus K'_{32} \qquad i = 31$$

where S'_i is the application of the S-box S_i mod 8 32 times in parallel, and L is the linear transformation.

Decryption algorithm

Decryption is different from encryption in that the inverse of the S-boxes must be used in the reverse order, as well as the inverse linear transformation and reverse order of the subkeys [25]. As in Figure 3.7 below:

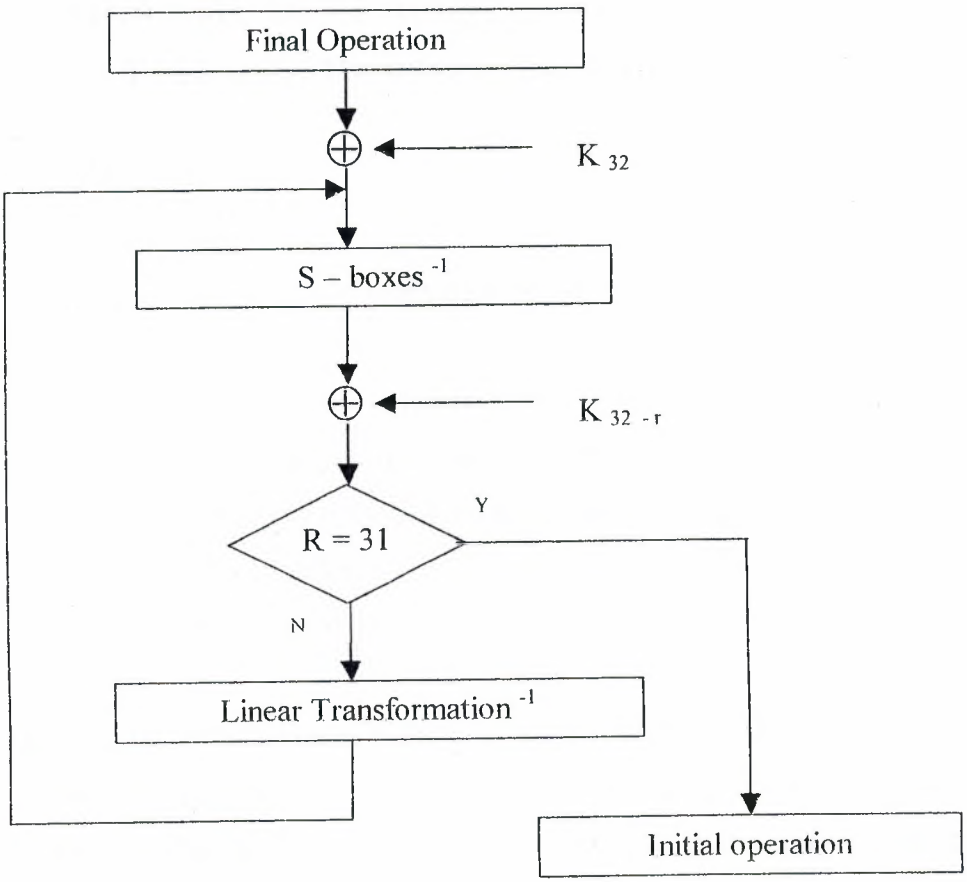

Figure 3.7 Decryption Structure

3.3.3. Construction of S-box

The S-boxes of Serpent are 4-bit permutations with the following properties: Each differential characteristic has a probability of at most ¼, and a one-bit input difference will never lead to a one-bit output difference. Each linear characteristic has a probability in the range ½ ± ¼ , and a linear relation between one single bit in the input and one single bit in the output has a probability in the range ½ ± $^1/_8$.The nonlinear order of the output bits as a function of the input bits is the maximum, namely 3. The S-boxes were generated in the following manner, which was inspired by RC4. Using a matrix with 32 arrays each with 16 entries. The matrix was initialized with the 32 rows of the DES S-boxes and transformed by swapping the entries in the rth array depending on the value of the entries in the (r+1) array and on an initial string representing a key. If the result array has the desired properties, save the array as a Serpent S-box. Repeat the procedure until 8 S-boxes have been generated. More formally, let serpent [.] be an array containing the least significant four bits of each of the 16 ASCII characters in the expression "sboxesforserpent". Let S-Box [.][.] be a (32 x 16)-array containing the 32 rows of the 8 DES S-Boxes, where S-Box[r][.] denotes the rth row. The function swap entries (. . .) is self-explanatory. The following pseudo-code generates the Serpent S-boxes.

```
index := 0
repeat
        currentsbox := index modulo 32
        for i:=0 to 15 do
                j := sbox [(currentsbox+1) modulo 32][serpent[i]]
                swapentries (sbox [currentsbox] [i],sbox [currentsbox][j])
        if sbox[currentsbox][.] has the desired properties, save it
        index := index + 1
until 8 S-boxes have been generated
```

In $Serpent_0$, using the DES S-boxes in order to inspire a high level of public confidence that it must had not inserted any trapdoor in them. A similar assurance for $Serpent_1$ comes from the fact that the S-boxes have been generated in this simple deterministic manner.

3.3.4. Key Generation

The input is pad of user key to 256 bits. After that it expanded to 33 128-bit subkeys $K_0, \ldots, K_{32}$, in the following way. Writing the key K as eight 32-bit words $w_{-8}, \ldots, w_{-1}$ and expand these to an intermediate key (which call prekey) $w_0, \ldots, w_{131}$ by the following affined recurrence:

$w_i := (w_{i-8} \oplus wi-5 \oplus w_{i-3} \oplus w_{i-1} \oplus \phi \oplus i) <<< 11$. Figure 3.8 below explains the key generation [12].

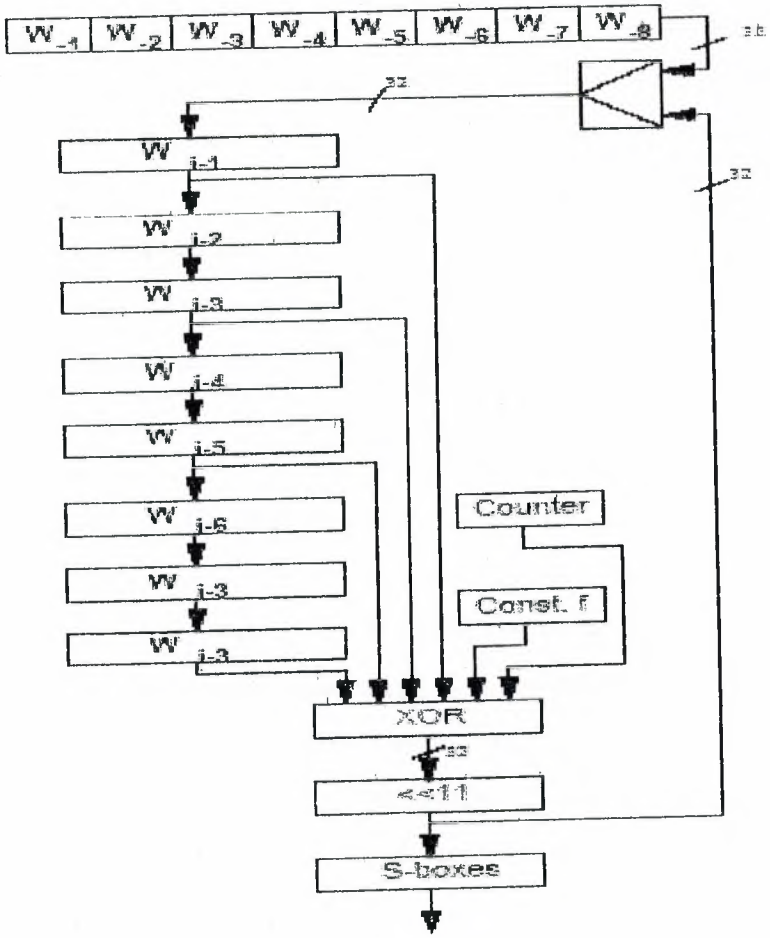

Figure 3.8 Serpent Key Generation

where ϕ is the fractional part of the golden ratio $(\sqrt{5} + 1) / 2$ or 0x9e3779b9 in hexadecimal. The underlying polynomial $x^8 + x^7 + x^5 + x^3 + 1$ is primitive, Which together with the addition of the round index is chosen to ensure an even distribution of key bits throughout the rounds, and to eliminate weak keys and related keys. The round keys are now calculated from the prekeys using the S-boxes, again in Bitslice mode.

The S-boxes is used to transform the prekeys w_i into words k_i of round key in the following way:

$$\{k_0, k_1, k_2, k_3\} := S_3(w_0, w_1, w_2, w_3)$$
$$\{k_4, k_5, k_6, k_7\} := S_2(w_4, w_5, w_6, w_7)$$
$$\{k_8, k_9, k_{10}, k_{11}\} := S_1(w_8, w_9, w_{10}, w_{11})$$
$$\{k_{12}, k_{13}, k_{14}, k_{15}\} := S_0(w_{12}, w_{13}, w_{14}, w_{15})$$
$$\{k_{16}, k_{17}, k_{18}, k_{19}\} := S_7(w_{16}, w_{17}, w_{18}, w_{19})$$

$$\{k_{124}, k_{125}, k_{126}, k_{127}\} := S_4(w_{124}, w_{125}, w_{126}, w_{127})$$
$$\{k_{128}, k_{129}, k_{130}, k_{131}\} := S_3(w_{128}, w_{129}, w_{130}, w_{131})$$

After that renumber the 32-bit values k_j as 128-bit subkeys K_i (for $i \in 2\ \{0,\ldots, r\}$) as follows: $K_i := (k_{4i}, k_{4i+1}, k_{4i+2}, k_{4i+3})$. Now apply IP to the round key in order to place the key bits in the correct column [25].

3.3.5. Security

As mentioned above, the initial version of Serpent used the DES S-boxes, as their differential and linear properties are well understood. The estimates indicated that the number of known/chosen plaintexts required for either type of attack would be well over 2^{100}. Having investigated how these S-boxes worked in our structure, It realized that it was simple to find S-boxes that would improve this to 2^{256}, and the aim to offer the best candidate algorithm led us to change to the S-boxes. By strength of 2^{256}, it means that a differential or linear attack against any key would take that many texts, assuming that they were available.

3.4. Twofish algorithm

3.4.1. Introduction

Developed by Bruce Schneier as a successor to his 64-bit Blowfish block cipher, John Kelsey, Doug Whiting, David Wagner, Chris Hall, and Niels Fergus on 5 June 1998. Twofish meets all the required NIST criteria, 128-bit block; 128, 192, and 256-bit key, efficient on various platforms, and some strenuous design requirements, performance as well as cryptographic. Twofish can be:

- Encrypt data at 285 clock cycles per block on a Pentium Pro, after a 12700 clock-cycle key setup.
- Encrypt data at 860 clock cycles per block on a Pentium Pro, after a 1250 clock-cycle key setup.
- Encrypt data at 26500 clock cycles per block on a 6805 smart card, after a 1750 clock-cycle key setup [8].
- No weak keys.
- Efficiency, both on the Intel Pentium Pro and other software and hardware platforms.
- Flexible design: e.g., accept additional key lengths, be implemental on a wide variety of platforms and applications; and be suitable for a stream cipher, hash function, and MAC. Simple design, both to facilitate ease of analysis and ease of implementation.

3.4.2. Encryption and decryption algorithms

Figure 3.9 shows an overview of the Twofish block cipher. Twofish uses a 16-r round Feistel-like structure with additional whitening of the input and output. The only non-Feistel elements are the 1-bit rotates. The rotations can be moved into the F function to create a pure Feistel structure, but this requires an additional rotation of the words just before the output-whitening step. The plaintext is split into four 32-bit words. In the input-whitening step, these are xor with four key words. Sixteen rounds follow this. In each round, the two words on the left are used as input to the g functions. The g function consists of four byte-wide key-dependent S-boxes, followed by a linear mixing step based on an MDS matrix. The results of the two g functions are combined using a Pseudo-Hadamard Transform (PHT), and two keywords are added. These two results are then xor into the words on the right (one of which is rotated left by 1 bit first, the other is rotated right afterwards). The left and right halves are then swapped for the next round. After all the rounds, the swap of the last round is reversed, and the four words are xor with four more key words to produce the ciphertext. More formally, the 16 bytes of plaintext $p_0, \ldots, p_{15}$ are first split into 4 words $P_0, \ldots, P_3$ of 32 bits each using the little-endian convention [].

$$P_i = \sum_{j=0}^{3} p(4i+j) \cdot 2^{8j} \qquad i=0, \ldots, 3$$

In the input-whitening step, these words are xor with 4 words of the expanded key.

$$R_{0,i} = P_i \oplus K_i \qquad i = 0, \ldots, 3$$

In each of the 16 rounds:

- The first two words are used as input to the function F, which also takes the round number as input.
- The third word is xor with the first output of F and then rotated right by one bit.
- The fourth word is rotated left by one bit and then xor with the second output word of F.
- Finally, the two halves are exchanged.

The previous step can be explain as:

$$(F_{r,0}\ F_{r,1}) = F(R_{r,0}, R_{r,1}, r)$$
$$R_{r+1,0} = ROR(R_{r,2} \oplus F_{r,0}, 1)$$
$$R_{r+1,1} = ROL(R_{r,3}, 1) \oplus F_{r,1}$$
$$R_{r+1,2} = R_{r,0}$$
$$R_{r+1,3} = R_{r,1}$$

For r = 0, ..., 15 and where ROR and ROL are functions that rotate their first argument left or right by the number of bits indicated by their second argument. The output-whitening step undoes the `swap' of the last round, and xor the data words with 4 words of the expanded key.

$$C_i = R_{16,(i+2) \bmod 4} \oplus K_{i+4} \qquad i = 0, \ldots, 3.$$

The four words of ciphertext are then written as 16 bytes $c_0, \ldots, c_{15}$ using the same little-endian conversion used for the plaintext [8]

$$C_i = [C_{[i/4]} / 2^{8\,(i \bmod 4)}] \bmod 2^8 \qquad i = 0, \ldots, 15.$$

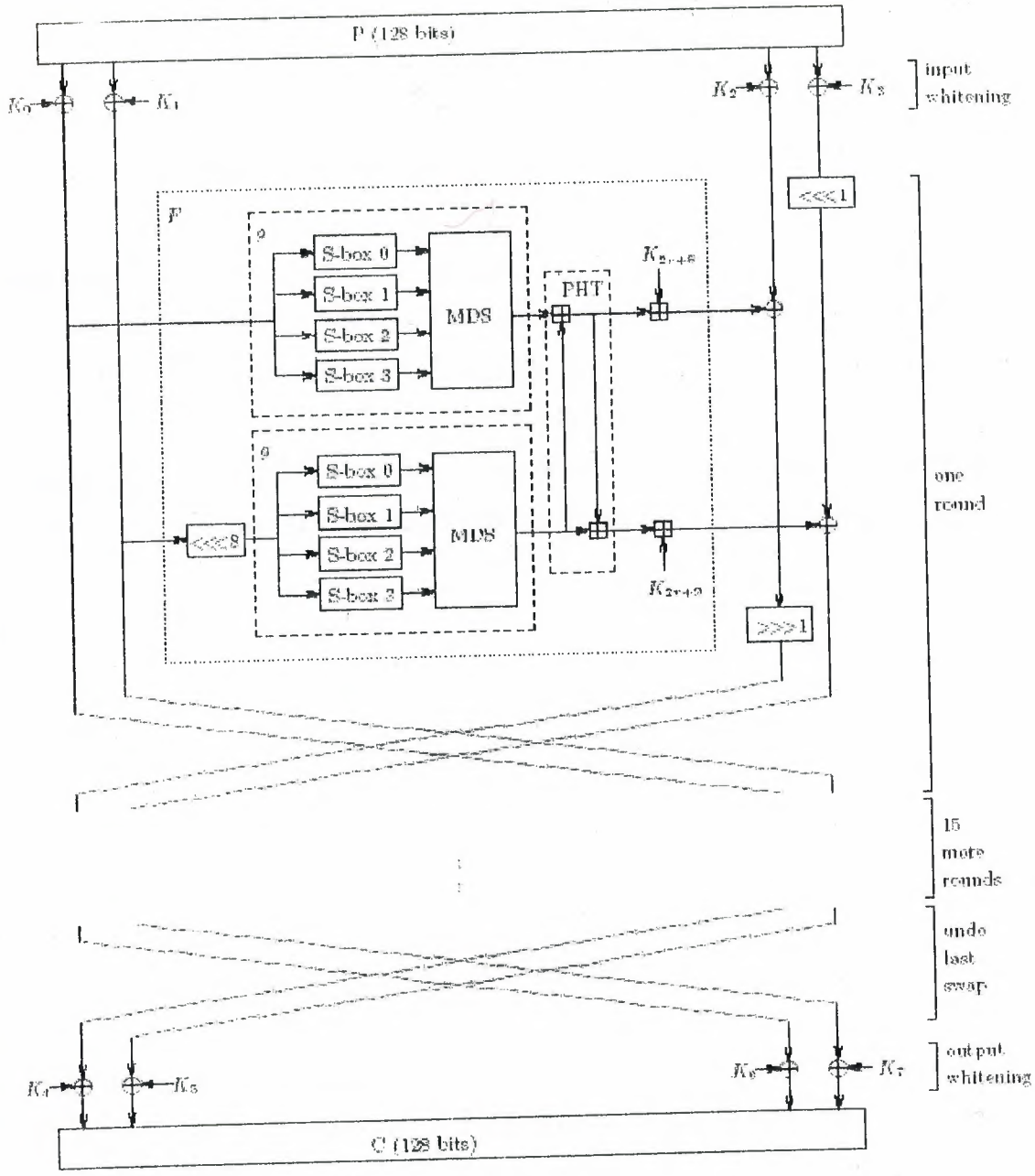

Figure 3.9 Encryption cipher of Twofish

The Function F

The function F is a key-dependent permutation on 64-bit values. It takes three arguments, two input words R_0 and R_1, and the round number r used to select the appropriate subkeys. R_0 is passed through the g function, which yields T_0. R_1 is rotated left by 8 bits and then passed through the g function to yield T_1. The results T_0 and T_1 are then combined in a PHT and two words of the expanded key are added.

$$T_0 = g(R_0)$$

$T_1 = g(ROL(R_1, 8))$

$F_0 = (T_0 + T_1 + K_{2r+8}) \bmod 2^{32}$

$F_1 = (T_0 + 2T_1 + K_{2r+9}) \bmod 2^{32}$

Where (F_0, F_1) is the result of F function.

The Function g

The function g forms the heart of Twofish. The input word X is split into four bytes. Each byte is run through its own key-dependent S-box. Each S-Box is objective, takes 8 bits of input, and produces 8 bits of output. The four results are interpreted as a vector of length 4 over $GF(2^8)$, and multiplied by the 4x4 MDS matrix (using the field $GF(2^8)$ for the computations). The resulting vector is interpreted as a 32-bit word, which is the result of g.

$$x_i = [X/2^{8i}] \bmod 2^8 \qquad i = 0, \ldots, 3$$

$$y_i = s_i[x_i] \qquad i = 0, \ldots, 3$$

$$\begin{pmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \end{pmatrix} = \begin{pmatrix} \\ \mathbf{MDS} \\ \\ \end{pmatrix} \cdot \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix}$$

$$zZ = \sum_{i=0}^{3} z_i . 2^{8i}$$

Where s_i are the key-dependent S-boxes and Z is the result of g. For this to be well-defined, it needs to specify the correspondence between byte values and the field elements of $GF(2^8)$. It represents $GF(2^8)$ as $GF(2)[x]/v(x)$, where $v(x) = x^8 + x^6 + x^5 + x^3 + 1$ is a primitive polynomial of degree 8 over GF(2). The field element $a = \sum_{i=0}^{7} a_i x^i$ with $a_i \in GF(2)$. The MDS matrix is given by:

$$\mathbf{MDS} = \begin{pmatrix} 01 & EF & 5B & 5B \\ 5B & EF & EF & 01 \\ EF & 5B & 01 & EF \\ EF & 01 & EF & 5B \end{pmatrix}$$

Where the elements have been written as hexadecimal byte.

3.4.3. Key Schedule

The key schedule has to provide 40 words of expanded key $K_0,\ldots, K_{39}$, and the 4 key-dependent S-boxes used in the g function. Twofish is defined for keys of length N = 128, N = 192, and N = 256. Padding them with zeroes until the next larger defined key length can use keys of any length shorter than 256 bits. Defining k = N=64. The key M consists of 8k bytes m0, …,m_{8k_1}. The bytes are first converted into 2k words of 32 bits each.

$$M_i = \sum_{j=0}^{3} m_{(4i+j)} \qquad i=0,\ldots, 2k-1.$$

And then into two word vectors of length k.

$$Me = (M_0, M_2, \ldots, M_{2k\_2})$$

$$Mo = (M_1, M_3, \ldots, M_{2k\_1})$$

A third word vector of length k is also derived from the key. This is done by taking the key bytes in groups of 8, interpreting them as a vector over GF(28), and multiplying them by a 4x8 matrix derived from an RS code. Each result of 4 bytes is then interpreted as a 32-bit word. These words make up the third vector.

$$\begin{pmatrix} S_{i,0} \\ S_{i,1} \\ S_{i,2} \\ Z_{i,3} \end{pmatrix} = \begin{pmatrix} \\ \mathbf{RS} \\ \\ \end{pmatrix} \cdot \begin{pmatrix} m_{8i} \\ m_{8i+1} \\ m_{8+2} \\ m_{8i+3} \\ m_{8i+4} \\ m_{8i+5} \\ m_{8i+6} \\ m_{8i+7} \end{pmatrix}$$

The RS matrix is given by:

$$\mathbf{RS} = \begin{pmatrix} 01\ A4\ 55\ 87\ 5A\ 58\ DB\ 9E \\ A4\ 56\ 82\ F3\ 1E\ C6\ 68\ E5 \\ 02\ A1\ FC\ C1\ 47\ AE\ 3D\ 19 \\ A4\ 55\ 87\ 5A\ 58\ DB\ 9E\ 03 \end{pmatrix}.$$

The three vectors Me, Mo, and S forms the basis of the key schedule.

3.5. MARS algorithm

3.5.1. Introduction

MARS is a shared-key block cipher, with a block size of 128 bits and a variable key size, ranging from 128 to over 400 bits. It was designed to meet and exceed the requirements for a standard for shared-key encryption in the next few decades. The main theme behind the design of MARS is to get the best security/performance tradeoff by utilizing the strongest tools and techniques available today for designing block ciphers. As a result, MARS provides a very high level of security, combined with much better performance than other existing ciphers. MARS offers better security than triple-DES. In particular, all the known cryptanalytical attacks (including linear and differential cryptanalysis) require more data than is available and hence these attacks are impossible against MARS. Also, the design principles of MARS make it likely that MARS would remain resilient even in the face of new cryptanalytical techniques [11].

3.5.2. Encryption and Decryption Algorithms

MARS takes as input four 32-bit data words. The cipher itself is word oriented, in that all the internal operations are performed on 32-bit words, and hence the internal structure is endian-neutral [32].

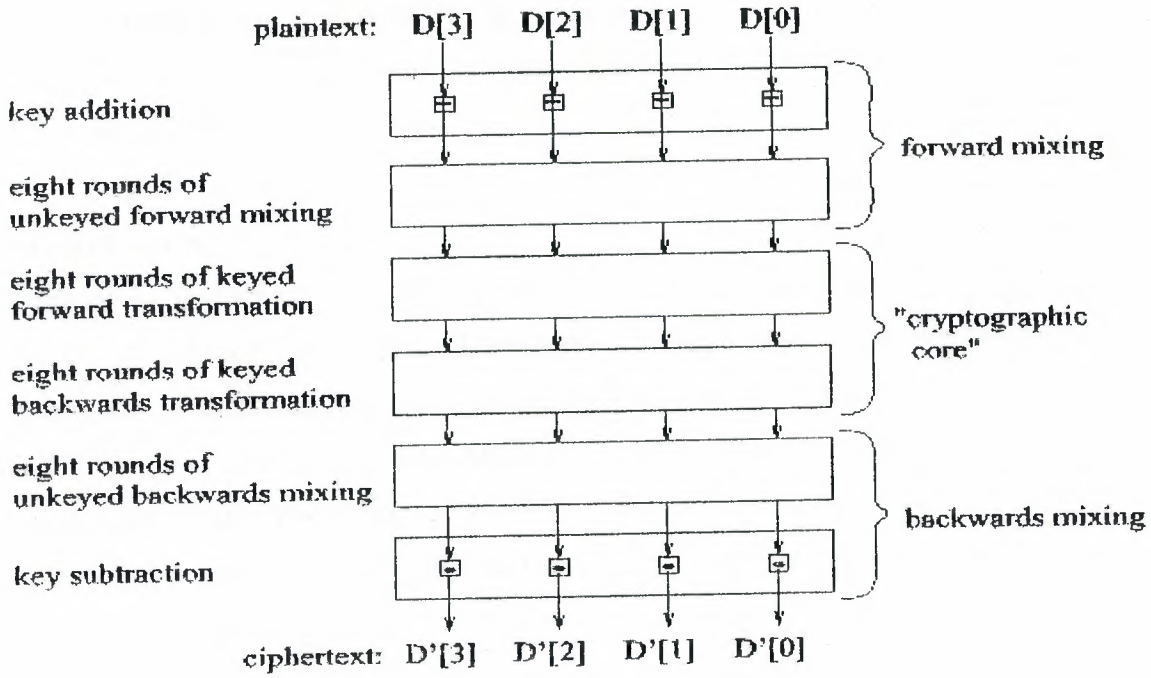

Figure 3.10 High-level Structure of Encryption.

High level structure

The general structure of the cipher is depicted in Figure 3.10 the cipher consists of a "cryptographic core" of keyed transformation, which is wrapped with two layers providing rapid key avalanche. Means the encryption structure consist of three phases:

- The first phase provides rapid mixing and key avalanche, to frustrate chosen-plaintext attacks, and to make it harder to "strip out" rounds of the cryptographic core in linear and differential attacks. It consists of addition of key words to the data words, followed by eight rounds of S-box based, unkeyed type-3 Feistel mixing.

- The second phase is the "cryptographic core" of the cipher, consisting of sixteen rounds of keyed type-3 Feistel transformation. To ensure that encryption and decryption have the same strength, the first eight rounds is performed in "forward mode" while the last eight rounds are performed in "backwards mode".

- The last phase again provides rapid mixing and key avalanche, this time to protect against chosen-ciphertext attacks. This phase is essentially the inverse of the first phase, consisting of eight rounds of the same type-3 Feistel mixing as in the first phase (except in "backwards mode"), followed by subtraction of key words from the data words.

Forward mixing

In this phase first thing is to add a key word to each data word, and then perform eight rounds of unkeyed type-3 Feistel mixing, combined with some additional mixing operations. In each round it need to use one data word (called the source word) to modify the other three data words (called the target words). The four bytes of the source word as indices into two S-boxes, S0 and S1, each consisting of 256 32-bit words, and xor or add the corresponding S-box entries into the other three data words. If it denote that the four bytes of the source words by b_0, b_1, b_2, b_3 (where b_0 is the lowest byte and b_3 is the highest byte), then using b_0, b_2 as indices into the S-box S0 and b_1, b_3 as indices into the S-box S1. First thing xor S0[b_0] into the first target word, and then add S1[b_1] to the same word. Also add S0[b_2] to the second target word and xor S1[b_3] to the third

target word. Finally, rotate the source word by 24 positions to the right. For the next round rotating the four words, so that the current first target word becomes the next source word, the current second target word becomes the next first target word, the current third target word becomes the next second target word, and the current source word become the next third target word. In addition, after each of four specific rounds it need to add one of the target words back into the source word. Specifically, after the first and fifth rounds adding the third target word back into the source word, and after the second and sixth rounds adding the first target word back into the source word. The reasons for these extra mixing operations are to eliminate some easy differential attacks against the mixing phase, to break the symmetry in the mixing phase and to get faster avalanche. Figure 3.11shows the forward mixing phase [11].

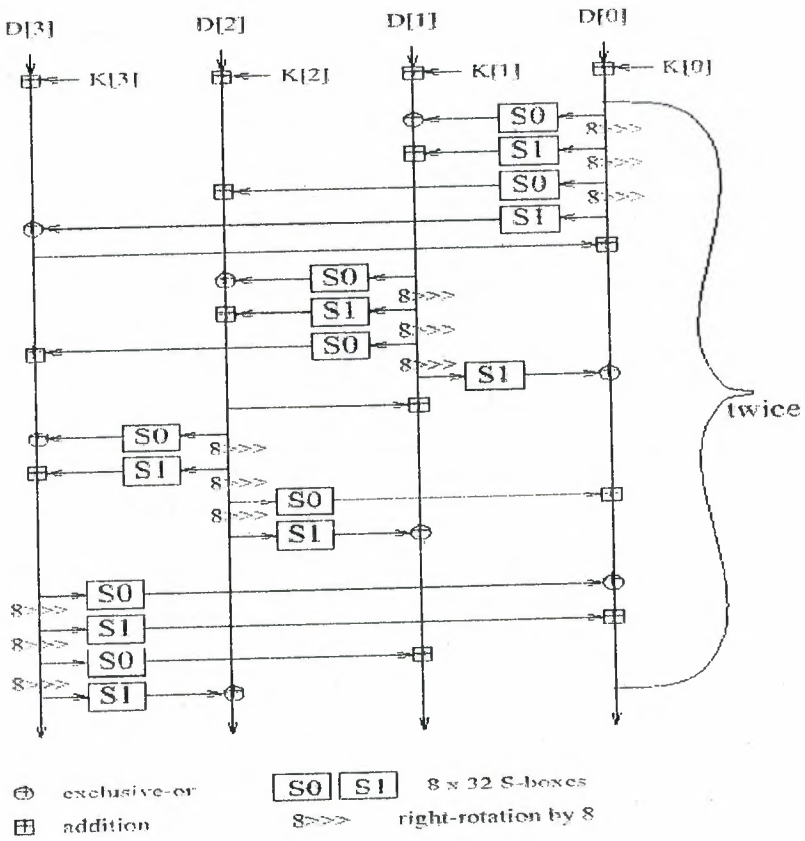

Figure 3.11 Structure of forward mixing phase.

Main keyed transformation

The "cryptographic core" of the MARS cipher is a type-3 Feistel network, consisting of sixteen rounds. In each round using a keyed E-function (E for expansion), which is

based on a novel combination of multiplication, data-dependent rotations, and an S-box lookup. This function takes as input one data word and returns three data words as output. The structure of the Feistel network is depicted in Figure 3.12, and the E-function itself is diagrammed in Figure 3.13. In each round using one data word as the input to the E-function, and the three output words from the E-function are added or xored to the other three data words. In addition, the source word is rotated by 13 positions to the left. To ensure that the cipher has the same resistance to chosen ciphertext attacks as it has for chosen plaintext attacks, the three outputs from the E-function are used in a different order in the first eight rounds than in the last eight rounds. Namely, in the first eight rounds adding the first and second outputs of the E-function to the first and second target words, respectively, and xor the third output into the third target word. In the last eight rounds, add the first and second outputs of the E-function to the third and second target words, respectively, and xor the third output into the first target word [13].

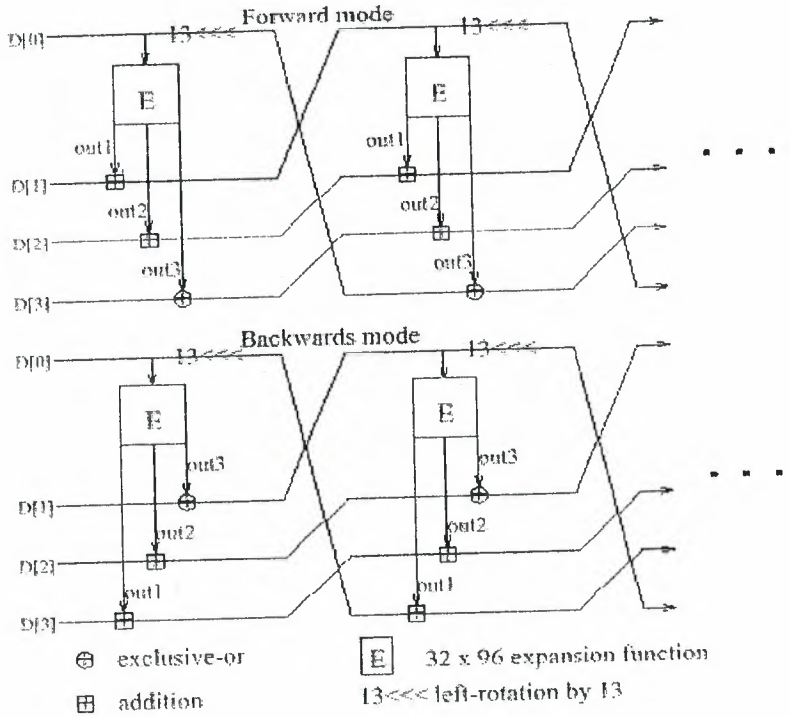

Figure 3.12 Type-3 Feistel network of main keyed transformation.

The E-function

The E-function takes as input one data word and uses two more key words to produce three output words. In this function using three temporary variables, denoted below by L, M and R (for left, middle and right). Below it also refer to these variables as the three

"lines" in the function. Initially, setting R to hold the value of the source word rotated by 13 positions to the left, and setting M to hold the sum of the source word and the first key word. After that view the lowest nine bits of M as an index to a 512-entry S-box S (which is obtained by concatenating S0 and S1 from the mixing phase), and set L to hold the value of the corresponding S-box entry. After that multiply the second key word (constrained to contain an odd integer) into R and then rotate R by 5 positions to the left (so the 5 highest bits of the product becomes the 5 lowest bits of R after the rotation). Then xoring R into L, and also view the five lowest bits of R as a rotation amount between 0 and 31, and rotate M to the left by this amount. Next, rotating R by 5 more positions to the left and xor it into L. Finally, again the five lowest bits of R view as a rotation amount and rotate L to the left by this amount. The first output word of the E-function is L, the second is M and the third is R [21].

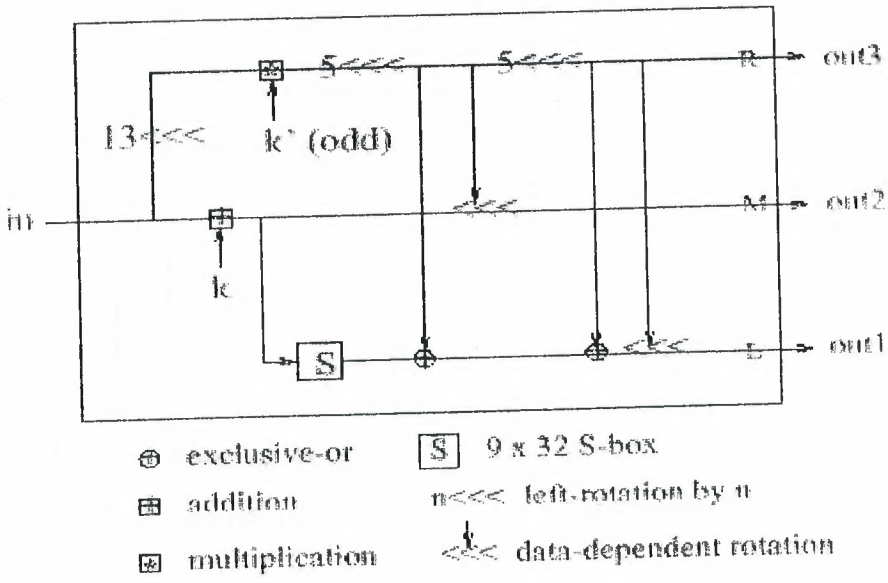

Figure 3.13 The E-function of the main keyed transformation.

Backwards mixing

The structure of the backward mixing showed in Figure 3.14 very similar to the one of the forward mixing. It consists also of 8 rounds of a unkeyed type-3 Feistel network, followed by a key subtraction step.

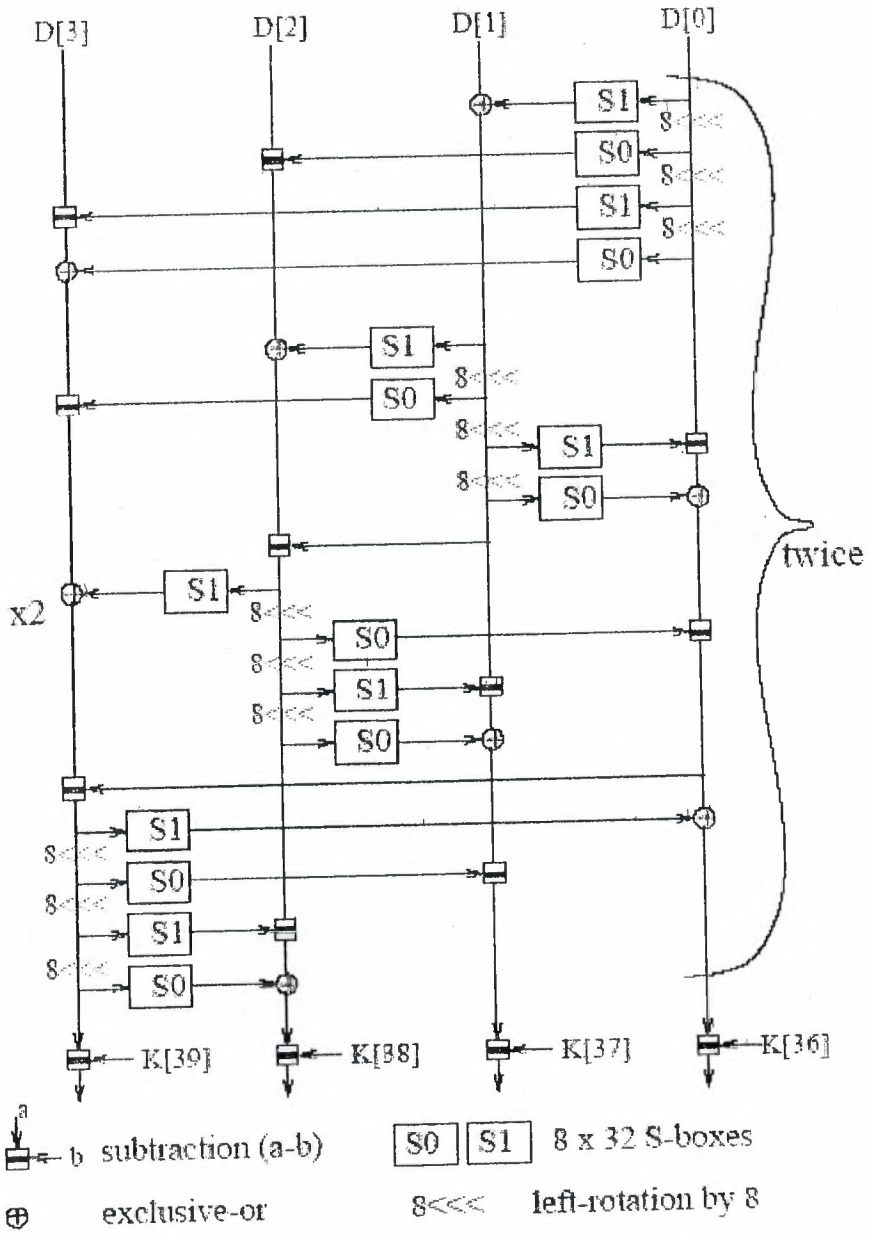

Figure 3.14 Structure of the backwards-mixing phase

The decryption process is the inverse of the encryption process. The code for decryption is similar (but not identical) to the code for encryption. to see more details about decryption [11].

3.5.3. Construction of S-box

In the design of the S-box S, generated the entries of S in a "pseudorandom fashion" and tested that the resulting S-box has good differential and linear properties. The

"pseudorandom" S-boxes were generated by setting for i= 0, …, 122, j= 0, ...,4. S[5i+j]= SHA^{-1}(5i| c1| c2| c3) (where SHA^{-1}(), is the ^{j}th word in the output of SHA^{-1}). Here i view as a 32-bit unsigned integer, and c1,c2,c3 are some fixed constants. In this thesis c=0xb7e15162, c2= 0x243f6a88 and varied c3 until S-box with good properties. SHA-1 as an operation on byte-streams, and use little-endian convention to translate between words and bytes. The properties of the S-box, which it tested as following:

Differential properties. Requiring that the S-box has the following properties:

(1) The S-box does not contain the all-zero or the all-one word.

(2) Within each of the two S-boxes S_0 and S_1 every two entries differ in at least three of the four bytes. (Note that it is very unlikely that a random S-box will have this property, and so first "fix" the S-box by modifying one of the entries in each pair that violates this condition).

(3) S does not contain two entries S[i], S[j]($i \neq j$)such that s[i]=s[j], s[i] = -s[j]

(4) S has (2^{512}) distinct xor-differences and 2 x (2^{512}) distinct subtraction-differences.

(5) Every two entries in S differ by at least four bits.

Linear properties. Trying to minimize the following quantities:

(6) Parity bias: $\left| \Pr_x[\text{parity}[S[x]=0] - \frac{1}{2} \right|$. Requiring that the parity bias of S be at most 1/30.

(7) Single-bit bias: $\forall j$, $\left| \Pr_x[\text{parity}[S[x]_j = 0] - \frac{1}{2}) \right|$. Requiring that the parity bias of S be at most 1/30.

(8) Two consecutive bits bias: $\forall j$, $\left| \Pr[\text{parity}[S[x]_j \oplus S[x]_{j+1} = 0] - \frac{1}{2}) \right|$. Requiring that the two-bit bias of S be at most 1/30.

(9) Single-bit correlation: $\forall i, j$ $\left| \Pr[\text{parity}(S[x]_j \oplus x_i = 0] - \frac{1}{2}) \right|$.

3.5.4. Key expansion

The key expansion procedure expands a given key array k[], consisting of n 32-bit words (where n is any number between 4 and 14) into an array K[] of 40 words. Noting

that the original key is not required to have any structure .In addition, the key expansion procedure also guarantees that the key words, which are used for multiplication in the encryption procedure, have the following properties

- The two lowest bits in a key word which is used for multiplication are set to1
- None of these key words contains ten consecutive 0's or ten consecutive 1's.

The procedure consists of the following steps

1. Initially, the original key material is copied into a temporary table T[ ] of 15 words, followed by the number of words n, and zeroes. Namely, setting
   T [0... n-1]=k [0... n-1], T [n]=n, T [n+1...14]=0.

2. Then, the following process is repeated four times, where each iteration computes the next ten words of the expanded key:

   (a) The array T[ ] is transformed using the following linear formula
   T [i] $\oplus$ ((T[i-7 mod 15] $\oplus$ T [i-2 mod 15]) <<< 3) $\oplus$ (4i+j),
   where j is the iteration number initialize by zero, and i from 0 to 14.

   (b) Next, stir the array T [] using four rounds of type-1 Feistel-network. Specifically, repeatting four times the operation.
   T [i] = (T [i] + S [low 9 bits of T [i-1 mod 15]]) <<< 9, where i= 0,1, ... 14.

   (c) Then take 10 of the words in T [] and reorder them into the next ten words of the expanded key array, K []. This is done by setting
   K [10j + i] = T [4i mod 15], i=0, 1, ...,9, and j is the iteration number.

3. Finally, going over the sixteen words, which are used in the cipher for multiplication (these are words K[5], k[7], ..., k[37]), and modify them to have the two properties from above. Noting that the probability that a randomly chosen word does not have the second property is about 1/41. Process each of the words K [5], k [7], ..., K[37] as follows:

i. Recording the two lowest bits of K[i], by setting j = K[i] 8 3 and then consider the word with these two bits set to 1, w = K[j] V 3.

ii. Constructing a mask M of the bits in w, which belong to a sequence of ten (or more) consecutive 0's or 1's. Namely, having M_L if and only if w_L belongs to a sequence of ten consecutive 0's or 1's. Then resetting to 0 the 1's in M that correspond to the "end-points of runs of 0's or 1's in w", and also the two lowest bits and the highest bit in M. More precisely, the ith bit of M is reset to 0 if i<2, i=31,or if the ith bit of W differs from either the (i+1)th or the(i-1)th bits. For example, assume that w = $0^3 1^{13} 0^{12} 1011$ (where by 0^i, 1^i denote as consecutive 0's or 1's, respectively). In this case the first set M = 0^3 1^{25} 0^4, and then reset the 1's in bit positions 4, 15,16 and 28 to get M= $0^4 1^1 0 0 1 1^{10} 0^5$.

iii. Next, using a fixed four-word table B to "fix w", where the four entries in B are chosen so that they (and their cyclic shifts) do not contain any seven consecutive 0's or ten consecutive 1's. Specifically, using the table B[ ] = f0xa4a8d57b, 0x5b5d193b, 0xc8a8309b, 0x73f9a978, (these are entries 265 through 268 in the S-box). The reason to choose these entries is that there are only 148-bit patterns, which appear twice in these entries (and their cyclic shifts), and no pattern appears more than twice. Using the two recorded bits j to select an entry from B, and use the lowest five bits of K[i-1]to rotate this.

iv. Entry = B[j] <<< (lowest 5 bits of K[i-1]).

v. Finally, then xoring the pattern p into w under the control of the mask M, and store the result in k[i], K[i] = w $\oplus$ (p8 M). Since the lowest two bits of M are 0's, then the lowest two bits of K[i] will be 1's (since those in re).also, the choice of B guarantees that K[i] will meet not have a sequence of ten consecutive 0's or 1's.

3.6. RC6 algorithm

3.6.1. Introduction

RC6™ is a new block cipher submitted to NIST for consideration as the new Advanced Encryption Standard (AES). The design of RC6 began with a consideration of RC5 [17] as a potential candidate for an AES submission. Modifications were then made to meet the AES requirements, to increase security, and to improve performance. Like RC5, RC6 is a fully parameterized family of encryption algorithms. A version of RC6 is more accurately specified as RC6-w/r/b where the word size is w bits, encryption consists of a nonnegative number of rounds r, and b denotes the length of the encryption key in bytes. Since the AES submission is targeted at w = 32 and r = 20, using RC6 as shorthand to refer to such versions. When any other value of w or r is intended in the text, the parameter values will be specified as RC6-w/r. Of particular relevance to the AES effort will be the versions of RC6 with 16, 24, and 32 byte keys.

3.6.2. Basic operation

For all variants, RC6-w/r/b operates on units of four w-bit words using the following six basic operations. The base-two logarithm of w will be denoted by lgw.

- a + b integer addition modulo 2^w
- a - b integer subtraction modulo 2^w
- a $\oplus$ b bitwise exclusive-OR of w-bit words
- a $\times$ b integer multiplication modulo 2^w
- a<<<b rotate the w-bit word a to the left by the amount given by the least significant lgw bits of b
- a>>>b rotate the w-bit word a to the right by the amount given by the least significant lgw bits of b

3.6.3. Key schedule

The key schedule of RC6-w/r/b is practically identical to the key schedule of RC5-w/r/b. The only difference is that more words are derived from the user-supplied key for use during encryption and decryption. To generate key schedule must follow these step

- The user supplies a key of b bytes. Sufficient zero bytes are appended to give a key length equal to a non-zero integral number of words,

- These key bytes are then loaded in little-endian fashion into an array of c w-bit words L[0],..., L[c - 1].

- Thus the first byte of key is stored as the low-order byte of L[0], ... , and L[c-1] is padded with high-order zero bytes if necessary.

- The number of w-bit words that will be generated for the additive round keys is (2r + 4) and these are stored in the array S [0, ..., 2r + 3].

- The constants P32 = B7E15163 and Q32 = 9E3779B9 are the same "magic constants" as used in the RC5 key schedule. The value of (P32) is derived from the binary expansion of (e – 2), where (e) is the base of the natural logarithm function. The value of Q32 is derived from the binary expansion of Φ -1, where Φ is the Golden Ratio. Figure 3.15 explains the key schedule [2].

Input: User-supplied b byte key preloaded into the c-word array L [0, ...,c - 1], r: number of round

Output: w-bit round keys S [0, ... 2r + 3]

Procedure:

S[0] = P_w

for i = 1 to 2r + 3 do

S[i] = S[i _ 1] + Q_w

A = B = i = j = 0

v = 3 × max (c, 2r + 4)

For s = 1 to v do

Begin

A = S [i] = (S [i] + A + B)<<<3

B = L [j] = (L [j] + A + B)<<<(A + B)

i = (i + 1)mod(2r + 4)

j = (j + 1)mod c

End

Figure 3.15 key schedules for RC6-w/r/b.

3.6.4. Encryption and decryption

RC6 works with four w-bit registers A, B, C, D which contain the initial input plaintext as well as the output ciphertext at the end of encryption. The first byte of plaintext or ciphertext is placed in the least-significant byte of A, the last byte of plaintext or ciphertext is placed into the most-significant byte of D. Using (A, B, C, D) = (B, C, D, A) to mean the parallel assignment of values on the right to registers on the left. Figure3.16 below explain the procedure of encryption operation [9].

Input: Plaintext stored in four w-bit input registers A, B, C, D
r: Number of rounds
w-bit round keys S[0,…., 2r + 3] which obtained by key schedule.

Output: Ciphertext stored in A, B, C, D

Procedure:

```
Begin
    Pre-whitening
    B = B + S [0]
    D = D + S [1]
    r-round iterations
    For i = 1 To r do
        Begin
            t = (B x (2B + 1))<<<lg w
            u = (D x (2D + 1))<<<lg w
            A = ((A ⊕ t)<<<u) + S [2i]
            C = ((C ⊕ u)<<<t) + S [2i+ 1]
            (A, B, C, D) = (B, C, D, A)
        End For
    Post-whitening
    A = A + S[2r + 2]
    C = C + S[2r + 3]
End
```

Figure 3.16 Encryption with RC6-w/r/b

RC6 consists of three parts, pre-whitening, r-round iterations of round function, and post-whitening, RC6 encryption algorithms support multiple data block sizes. 128-bit RC6 encryption supports 20 rounds with each round using 2 round keys. 4 additional keys are used during pre and post rounds. Each round of RC6 encryption uses two multiplications, two additions, two exclusive-or operations, two fixed rotations and two variable rotations. Now the block diagram that explains encryption algorithm operation is show below in Figure 3.17 [18].

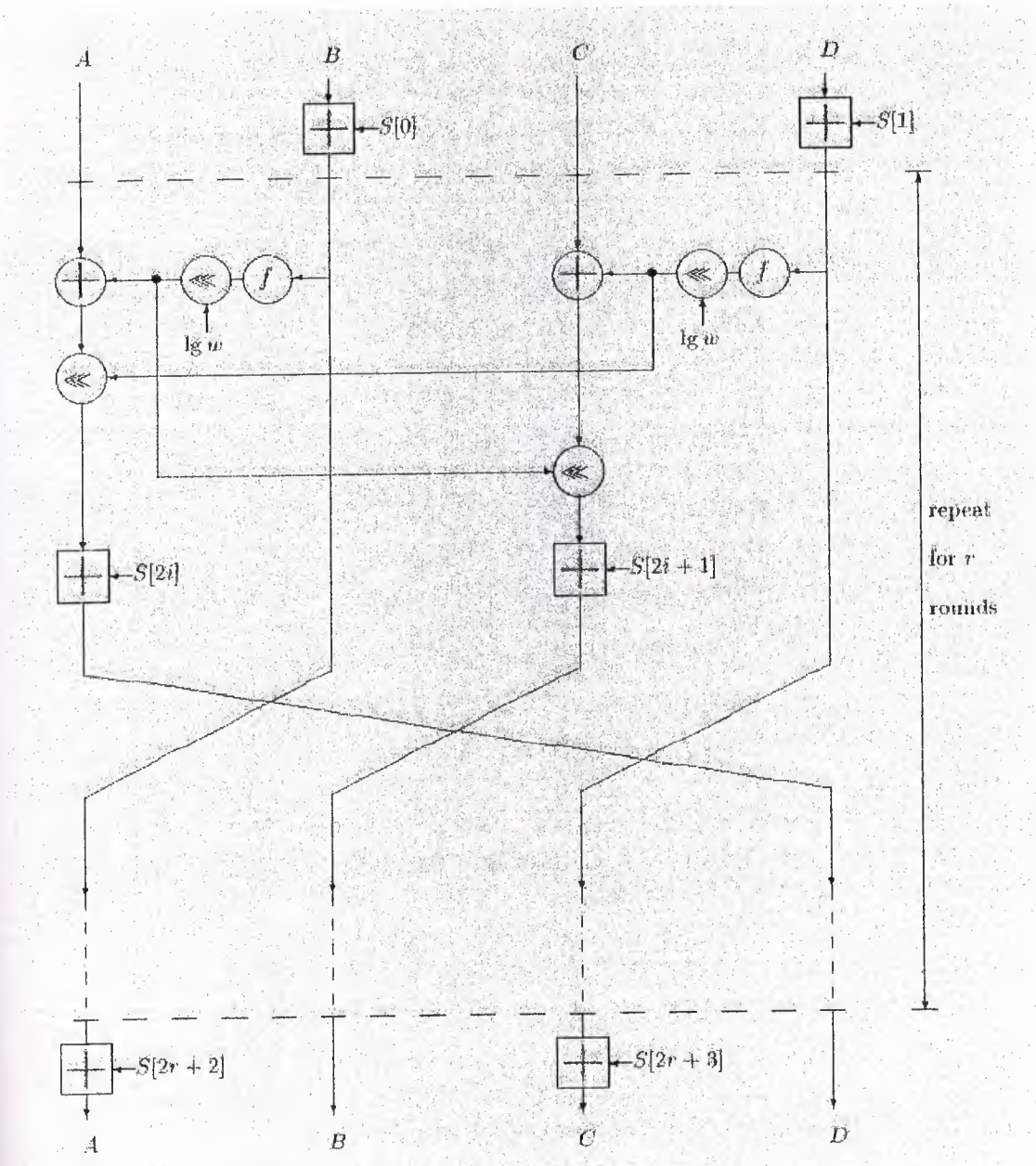

Figure 3.17 Encryption with RC6-w/r/b.

Note that the function f used in Figure 3.17 is f (x) = (x) x (2x + 1). Suppose the parameter to this function is register D then f (D) = D x (2D + 1). For decryption operation is opposite of encryption operation so the Figure 3.18 explain the procedure of decryption operation [9].

Input: Ciphertext stored in four w-bit input registers A, B, C, D.
Number r of rounds
w-bit round keys S[0,…., 2r + 3]

Output: Plaintext stored in A, B, C, D

Procedure:

```
Begin
    Pre-whitening
    C = C – S [2r + 3]
    A = A – S [2r + 2]
    R-round iterations
    For i = r down To 1 do
    Begin For
        (A, B, C, D) = (D, A, B, C)
        u = (D x (2D + 1))<<<lg w
        t = (B  x (2B + 1))<<<lg w
        C = ((C - S[2i + 1])>>>t) ⊕  u
        A = ((A – S [2i])>>>u) ⊕ t
    End For
    Post-whitening
    D = D – S [1]
    B = B – S [0]
End
```

Figure 3.18 Decryption with RC6-w/r/b.

3.6.5. Security and simplicity

During the design of RC6 the following considerations were uppermost Security, simplicity, good performance. The simplicity of RC5 has made it an attractive object for research. By being readily accessible to both crude and sophisticated analysis many people have been encouraged to look at the cipher and to assess the security it offers. RC6 was designed to build on the experience gained in using RC5 and to build on the security offered by a remarkably simple cipher. One can view the design of RC6 as progressing through the following steps:

- Start with the basic half-round loop of RC5:

```
for i = 1 to r do
{
    A = ((A⊕ B)<<<B) + S [i]
    (A, B) = (B, A)
}
```

- Run two copies of RC5 in parallel: one on registers A, B and one on registers C, D.

```
For i = 1 to r do
{
    A = ((A⊕ B)<<<B) + S[2i]
    C = ((C⊕ D)<<<D) + S[2i+ 1]
    (A, B) = (B, A)
    (C, D) = (D, C)
}
```

- At the swap stage, instead of swapping A with B and C with D, permute the registers by (A, B, C, D) = (B, C, D, A), so that the AB computation is mixed with the CD computation. At this stage the inner loop looks like:

```
For i = 1 to r do
{
    A = ((A ⊕ B)<<<B) + S [2i]
```

C = ((C⊕ D)<<<D) + S [2i+ 1]

(A, B, C, D) = (B, C, D, A)

}

- Mix up the AB computation with the CD computation further, by switching where the rotation amounts come from between the two computations:

```
for i = 1 to r do
{
   A = ((A⊕ B)<<<D) + S [2i]
   C = ((C ⊕ D)<<<B) + S [2i+ 1]
   (A, B, C, D) = (B, C, D, A)
}
```

- Instead of using B and D in a straightforward manner as above, using transformed versions of these registers, for some suitable transformation. Our security goals are that the data-dependent rotation amount that will be derived from the output of this transformation should depend on all bits of the input word and that the transformation should provide good mixing within the word. The particular choice of this transformation for RC6 is the function $f(x) = x(2x + 1)(\text{mod } 2^w)$ followed by a left rotation by five bit positions. This transformation appears to meet our security goals while taking advantage of simple primitives that are efficiently implemented on most modern processors. Note that f(x) is one-to-one modulo 2w, and that the high-order bits of f(x), which determine the rotation amount used, depend heavily on all the bits of x. See The Security of the RC6™ Block Cipher .

```
for i = 1 to r do
{
   t = (B x (2B + 1))<<<5
   u = (D x (2D + 1))<<<5
   A = ((A⊕ t)<<<u) + S[2i]
   C = ((C⊕ u)<<<t) + S[2i+ 1]
```

(A, B, C, D) = (B, C, D, A)

}

- At the beginning and end of the r rounds, add pre-whitening and post-whitening steps. Without these steps, the plaintext reveals part of the input to the first round of encryption and the ciphertext reveals part of the input to the last round of encryption. The pre- and post-whitening steps help to disguise this and leaves us with RC6:

```
B = B + S[0]
D = D + S[1]
for i = 1 to r do
{
   t = (B x (2B + 1))<<<5
   u = (D x (2D + 1))<<<5
   A = ((A⊕ t)<<<u) + S[2i]
   C = ((C⊕ u)<<<t) + S[2i+ 1]
   (A, B, C, D) = (B, C, D, A)
}
      A = A + S [2r + 2]
      C = C + S [2r + 3]
```

- While it might appear that the evolution from RC5 to RC6 was straightforward, it in fact involved the design and analysis of literally dozens of alternatives. RC6 is the design that captures the spirit of our three goals of security, simplicity and performance the most effectively. Note that in the preceding development, the decision to expand to four 32-bit registers was made first (for performance reasons), and then the decision to use the quadratic function f(x) = x(2x + 1)(mod 2w) was made later. If you had decided to stick with a two register version of RC6 then you might have had the following encryption scheme as an intermediate:

```
B = B + S[0]
for i = 1 to r do
```

{

t = B x (2B + 1)<<<5

A = ((A ⊕ t)<<<t) + S [i]

(A, B) = (B, A)

}

A = A + S[r + 1]

This variant of RC6 may be of independent interest, particularly when sup-Port for 64-bit arithmetic in C improves. However merely mention this as and aside here.

3.6.6. Good performance for a given level of security

Two significant changes in RC6 are the introduction of the quadratic function B x (2B + 1) and the fixed rotation by five bits. The quadratic function is aimed at providing a faster rate of diffusion thereby improving the chances that simple differentials will spoil rotation amounts much sooner than is accomplished with RC5. The quadratically transformed values of B and D are used in place of B and D to modify the registers A and C, increasing the nonlinearity of the scheme while not losing any entropy (since the transformation is a permutation). The fixed rotation by five bits plays a simple yet important role in complicating both linear and differential cryptanalysis.

3.6.7. Security of RC6

Conjecturing that to attack RC6 the best approach available to the cryptanalyst is that of exhaustive search for the b-byte encryption key (or the expanded key array S [0, ..., 43] when the user-supplied encryption key is particularly long). The work effort required for this is min (2^{8b}, 2^{1408}) operations. Don Coppersmith observes, however, that at the expense of considerable memory and off-line pre-computation one can mount a meet-in-the-middle attack to recover the expanded key array S [0, ...,43]. This would require 2^{704} on-line computations and so the work effort required to recover the expanded key array might best be estimated by min (2^{8b}, 2^{704}) operations. The more advanced attacks of differential and linear cryptanalysis, while being feasible on small-round versions of the cipher, do not extend well to attacking the full 20-round RC6 cipher. The main difficulty is that it is hard to find good iterative characteristics or linear approximations

with which an attack might be mounted. It is an interesting challenge to establish the most appropriate goals for security against these more advanced attacks. To succeed, these attacks typically require large amounts of data, and obtaining 2^a blocks of known or chosen plaintext-ciphertext pairs is a very different task from trying to recover one key from among 2^a possibilities. It is worth observing that with a cipher running at the rate of one terabit per second (that is, encrypting data at the rate of 10^{12} bits/second), the time required for 50 computers working in parallel to encrypt 2^{64} blocks of data is more than a year; to encrypt 2^{80} blocks of data is more than 98, 000 years, and to encrypt 2^{128} blocks of data is more than 10^{19} years. While having a data requirement of 2^{64} blocks of data for a successful attack might be viewed as sufficient in practical terms, the community as a whole will decide which level of security a cipher, in particular AES candidates should satisfy. Should this be less than a data requirement of 2^{128} blocks of data then the number of rounds of RC6 could potentially be reduced from our initial suggestion of 20 rounds, thereby providing an improvement in performance. For attacking an eight-round version of the cipher, RC6-32/8/b, one can construct six-round characteristics or linear approximations. Assuming that these could be used to attack the eight-round version of the cipher the estimated data required to mount a differential cryptanalytic attack on RC6-32/8/b would be around 2^{56} chosen plaintext pairs, and to mount a linear cryptanalytic attack would be around 2^{47} known plaintexts. This includes some consideration of more sophisticated phenomena such as differentials and linear hulls, but we might still expect more customized techniques to reduce these figures by a moderate amount. However they provide a reasonable illustration of the security that might be offered by a version of RC6 with a few rounds. Currently, it seems that a differential attack on the full 20-round RC6 cipher appears to be most easily accomplished by using a six-round iterative characteristic together with some customized beginning and ending characteristics. Considering a variety of options, the probability of one of the best 18-round characteristics we are aware of in attacking RC6 is around 2^{-238} and uses integer subtraction as the notion of difference. To use this characteristic in an attack would require more than the total number of available chosen plaintext/ciphertext pairs. While we expect the amount of data required for an attack to drop as more detailed analysis takes place we do not believe that differential cryptanalysis can be successfully applied to RC6. To mount a linear cryptanalytic attack, there appear to be two different options. The first might be to find a linear

approximation over several rounds that use a linear approximation across the quadratic function. Since there appear to be some very suitable linear approximations using the least significant bits of this function, this might be an appealing strategy. Indeed, one can establish useful six-round iterative linear approximations that can, at least in principle, be used to attack reduced-round versions of RC6. However, the bias of these approximations drops rapidly as more rounds are added, and soon the amount of data required for a successful attack exceeds the amount of data available. Instead, we note that an attacker might well pursue an alternative approach. It is possible to find a two-round iterative linear approximation that does not use an approximation across the combination of the quadratic function and fixed rotation by five bit positions. Using basic but established techniques to predict the bias of such an approximation, it will be observe that the data requirement to exploit this approximation over a version of RC6 with 16 rounds is about 2^{142} known plaintexts. Further analysis suggests that additional techniques might potentially be used to bring the data requirements down to a little under 2^{128} known plaintexts. This provided our rationale for choosing 20 rounds for RC6. With our current knowledge, the most successful avenue for a linear cryptanalytic attack on RC6 would be to use the two-round iterative approximation it have just mentioned to build up an 18-round linear approximation with which to attack the cipher. Using the same techniques as before to predict the data requirements to use this approximation at first sight it might need 2^{182} known plaintexts, an amount which exceeds the available data. Enhanced techniques might be useful in reducing this figure by a moderate amount (a pessimistic view suggests that such reductions would still leave an attack requiring 2^{155} known plaintexts) but in the final assessment we believe that the number of known plaintexts needed to exploit this approximation readily exceeds the maximum number of plaintexts available. We conclude that a linear cryptanalytic attack against RC6 is not possible using these techniques. Further, we believe that the use of more sophisticated techniques are exceptionally unlikely to provide sufficient gains as to offer attack requiring less than 2^{128} known plaintexts [9].

3.6.8. Flexibility and Future Directions

As it has already observed RC6 provides the user with a great amount of flexibility with regards to the size of the encryption key, the number of rounds and the word size of the basic computational unit. While the submission of RC6 for consideration as the

forthcoming AES is based around the use of 32-bit words (giving a block size of 128 bits), future developments and market demand might encourage an extension of RC6 to other block sizes. Of most importance may be block sizes of 256 bits, which would take advantage of a word size of 64 bits and the performance offered by the next generation of system architectures. Noting that further that the structure of RC6 allows one to exploit a certain degree of parallelism in the encryption and decryption routines. For example, the computation of t and u at each round can be computed in parallel, as can the updates of A and C. As processors move to include an increasingly amount of internal parallelism (e.g., with the move to superscalar architectures), implementations of RC6 should show increased throughput.

3.7. Comparison between five candidates algorithms for AES

3.7.1. General differences

The Table 3.4 shows the main differences between the five candidates algorithms for AES.

Table 3.4 General Differences

Properties	Algorithms				
	Rijndael	**Serpent**	**Twofish**	**RC6**	**MARS**
Core structure	Substitution Permutation Network	Substitution Permutation Network	Feistel Network	Feistel Network	Feistel Network
Design Principles	Modified Substitution Permutation Network – Square	Modified Substitution Permutation Network– Bitslice	Modified Blowfish	Modified RC5	Mixed structure DES
Design by	Joan Daemen and Vincent Rijmen	Ross Anderson, Eli Biham, and Lars Knudsen	Bruce Schneider	Professor Ronald L. Rivest, Sidney, and T. Yin	Carolynn Burwick, Don Coppersmith, Edward D'Avignon, IBM Corporation.

Number of round	10, 12, 14 rounds	32 round	16 rounds	20 rounds	32 round
Key length	128, 192, or 256 bit key	128, 192, 256 bit key	a variable-length key (128-, 192-, or 256-bit key)	128, 192, 256 bit key	Variable key size
Block Size	128-bit block cipher	128-bit block cipher	128-bit block cipher	128-bit block cipher	128-bit block cipher
Year Of Design	September 3, 1999	24th March 2000	15 June 1998	August 20, 1998	September, 22 1999
Minimal Secure Round	10 round	17 round	12 round	20 round	20 round

3.7.2. Operational differences

The Table 3.5 below shows the difference, which belongs to operation used by each algorithm.

Table 3.5 Operational Differences

Algorithms	**Operation used**				
Rijndael	S-box	Fixed rotation	x($GF(2^8)$)	⊕ (Key)	---
Serpent	⊕ (Key)	S-box	⊕ (Linear Transformation)	Permutation	Mixing operation
Twofish	S-box	x($GF(2^8)$)	X(mod 2^{32})	+(mod 2^{32})(Key)	⊕
RC6	x(mod 2^{32})	5-bit rotation	Xor(⊕)	Variable rotation	+(mod 2^{32})(key)
MARS	S-box	Fixed rotation	⊕	Data depend rotation	---

3.8.3. Encryption and Decryption times Difference

Table 3.6 compares the speed of encryption of five candidates for AES on Pentium Pro/II. The Table below explains the RC6 is very faster when byte is increase.

Table 3.6 Clock Cycles, per Byte, to encrypt different text sizes

Text size (bytes)	Algorithms				
	Rijndael	**Serpent**	**Twofish**	**RC6**	**MARS**
16	53	193	132	118	246
32	36	125	93	67	133
64	27	90	73	41	76
128	23	73	64	28	48
256	20	65	48	22	34
512	19	61	33	19	27
2^{10}	19	58	25	17	24
2^{11}	18	57	20	16	22
2^{12}	18	57	18	16	21
2^{13}	18	57	17	16	20
2^{14}	18	56	17	16	20

3.8. Summary

It is fair to say that all five of the finalist algorithms very strong cipher algorithms. They are all symmetric ciphers, similar to AES in terms of block size and key lengths. RC6™ appears to be the best choice for the AES-FIPS. It scores maximum points because of its proven security, simplicity implementation. Its scaleable performance characteristics make it more flexible for the different demanding environments [14].

4. RC6™ IMPLEMENTATION USING VISUAL C++ 6.0

4.1. Overview

By the times software for the encryption algorithms become easy to decrypt by attacker, may be because of weakness of algorithms or attacker uses development equipments or devices. For that we need algorithm that depends on strong computation and long key, so the suitable type of cryptography algorithms for that is single-key cryptography. Especially the five advanced algorithm for AES. This chapter describes the implementation of RC6™ and transmits data through the secure channel.

4.2. Application Requirements

The application works under the following requirements

4.2.1. Visual C++ 6.0

Why using Visual C++ version 6.0

- Visual C++ version 6.0 is a completely self-contained environment for creating, compiling, linking, and testing windows program.

- Visual C++ version 6.0 a range of fully integrated tools designed to make the whole process of writing windows program easy.

- Visual C++ version 6.0 includes two important tools which work in a wholly integrate way to help you write windows program. These are the AppWizard and the ClassWizard.

- The editor provides an interactive environment for creating and editing C++ source code. As well as usual facilities, such as cut and past.

- The compiler converts your source code into machine language, and detects and reports error in the compilation process. The compiler can detect a wide range of error s that are due to the invalid program code [33].

4.2.2. ActiveSkine 4.3

ActiveSkin is a powerful ActiveX control that changes the visual appearance of forms or dialogs, providing developers with full-featured support for 'application skins', or 'application look and feel'. ActiveSkin makes it easy to create programs with visually stunning, fully interactive user interfaces. The interfaces can even be created or edited directly at runtime. With ActiveSkin you could make your program look exactly like it

was from another OS, or create your own user WinAmp-like "skinned" interfaces, or just give it a different, distinctive look.

ActiveSkin supports features such as irregular or layered windows with animated shapes, soft shadows, animation, up to 50 transition effects, skin compression, hue adjusting, procedural textures, the ability to build any number of the skins directly into the application executable, a rich collection of SkinObjects (SkinForm, SkinFreeForm, SkinButton, SkinStatic, SkinTab, SkinFrame, Popup menus and more) that can be used in windowed and windowless mode and many more.
The ActiveSkin package also includes a selection of free skins and the "SkinBuilder" application, a visual skin editor that can be used to easily modify, preview or create skins.
ActiveSkin features:

- Full support for standard skins makes it possible to skin the entire application with a single line call

- Advanced application-specific skins support enables you to create stunning skins for your application only

- Full support of translucent layered windows for soft shadows, glows or glass-like effects

- Animated, interactive skins can be created entirely in SkinBuilder, without any additional coding needed

- Improved performance and reduced memory usage
- Lots of skins, that are available for royalty-free use to ActiveSkin customers
- Fade-In transitions make it possible to create splash screens
- Up to 50 built-in transition effects, alpha-blending operations, hue transforms and other image effects.
- Compiled skins, that can be built directly into executable
- An easy to use yet powerful SkinBuilder application

The ActiveSkin component is compatible with:

- Microsoft Windows 9x, ME, 2000 & XP
- Microsoft Visual Basic 5-6
- Microsoft Visual C++ 5-6
- Inprise/Borland Delphi 3-6
- Inprise/Borland C++ Builder 3-5 [35].

4.2.3. Network

This application sends ad receives files through network, so we need at least tow computer with this requirement:

- Microsoft Windows 9x, ME, 2000, and XP because Visual C++ for building 32-bit application. So each computer must satisfy these following requirements:
    1. At least a 486 Dx4 with 32-bit Mb of memory for windows NT.
    2. At least a 486 CPU and minimum 16Mb of memory for windows 9x.
    3. SVGA monitor.

4.3. Flow Chart Of Application0

The below Figure 4.1 describes the general structure of my application that consist of flowing structures:

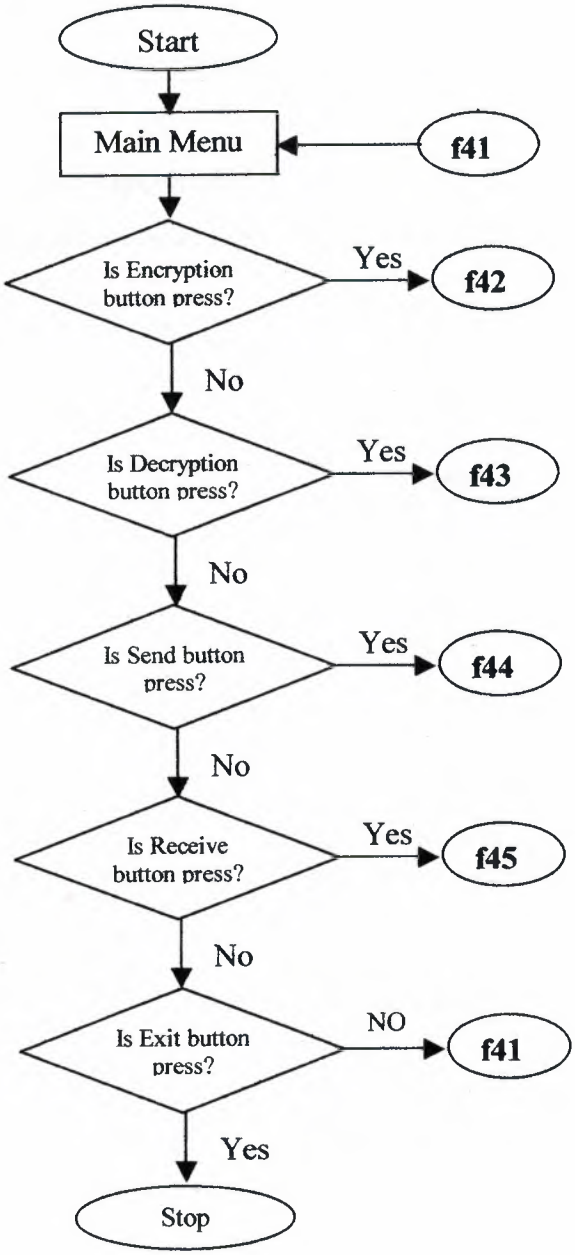

Figure 4.1 Main Flow Diagram

- Encryption operations appear when the decryption Button is pressed which will illustrated in Figure 4.2
- Decryption operations appear when the decryption Button is pressed which will illustrated in Figure 4.3

- Sending file operations appear when the sending Button is pressed which will illustrated in Figure 4.4
- Receiving file operations when the receiving Button is pressed which will illustrated in Figure 4.5
- Finally the programs will be exit when the exit Button is pressed.

4.3.1. Encryption Algorithm

The below diagram show in Figure 4.2 show the operation or the step of encryption.

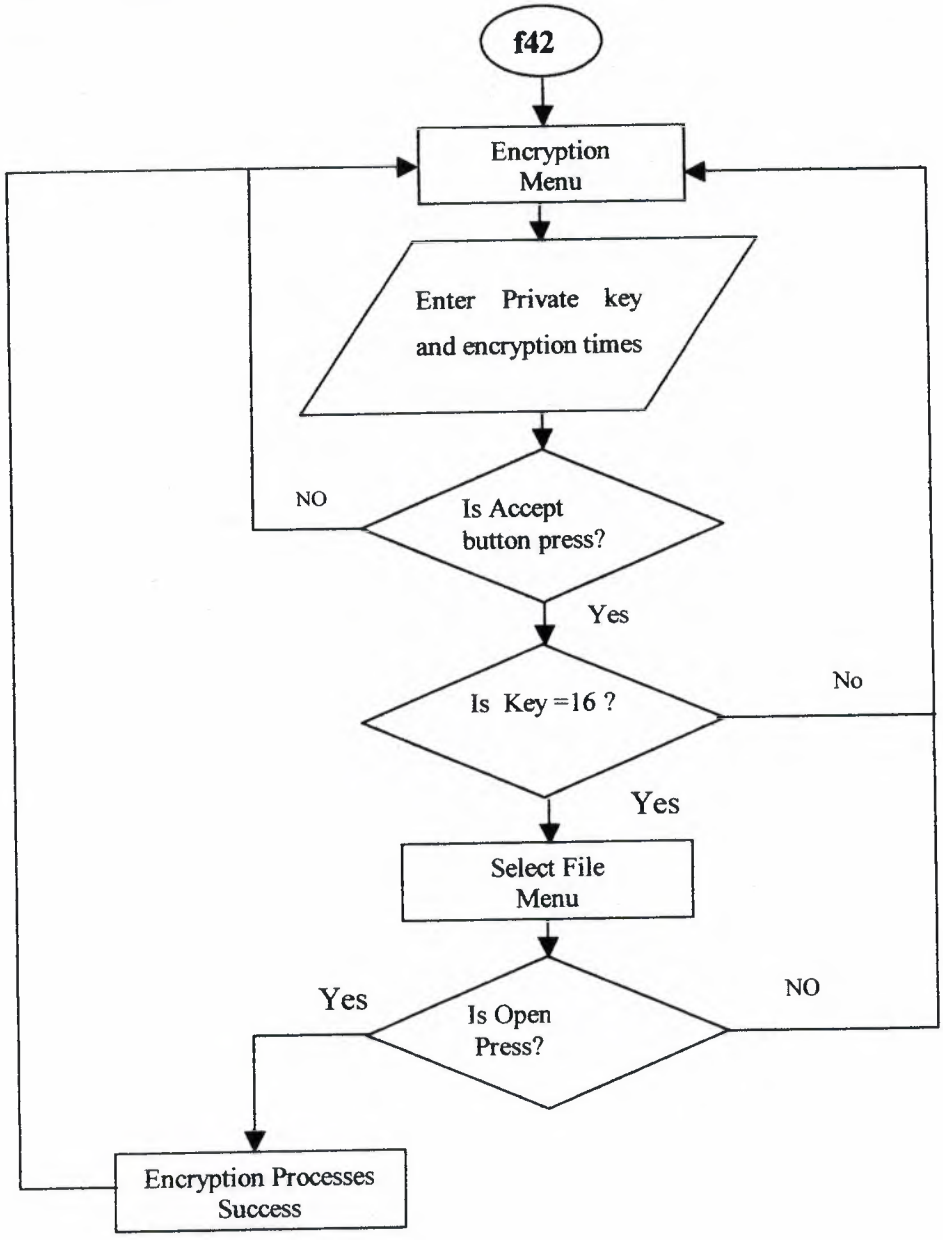

Figure 4.2 Encryption flow diagram

First thing you must enter the private key and encryption times. The length for private key is 16 characters (128-bit), which is the length of RC6™ . then if you press accept and the length is correct after that selecting file that will encrypt and then apply the RC6 encryption algorithm that described in chapter three then the encryption operation finish. But if the length of key is not correct or when selecting file menu you pres cancel you must begin from the first.

4.3.2. Decryption Algorithm

The below diagram show in Figure 4.3 show the operation or the step of decryption algorithms.

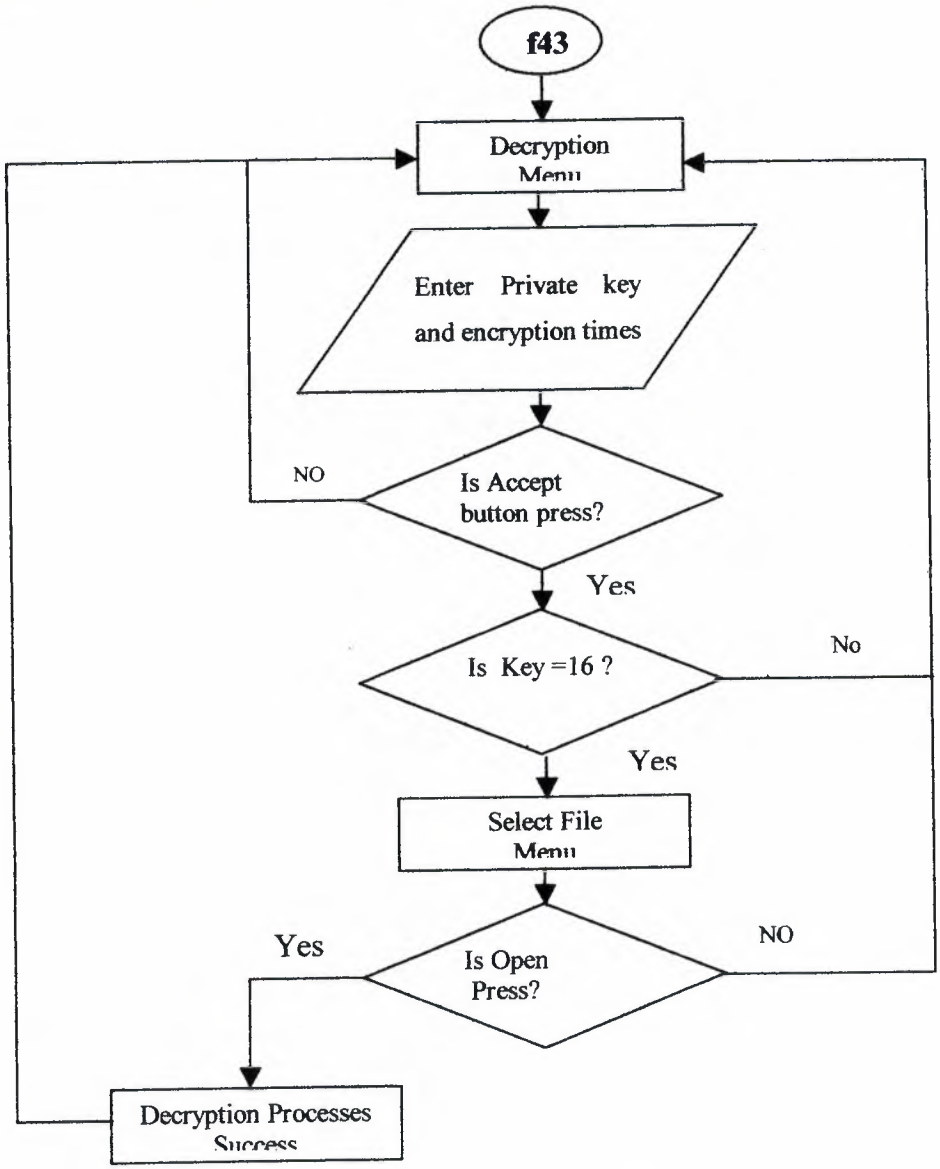

Figure 4.3 Decryption flow diagram

First thing you must enter the private key and decryption times. The length for private key is 16 characters (128-bit), which is the length of RC6™ . then if you press accept and the length is correct after that selecting file to decrypt it and then apply the RC6 decryption algorithm that described in chapter three then the decryption operation will be finish. But if the length of key is not correct or when selecting file menu you pres cancel you must begin from the first.

4.3.3. Sending Algorithm

The below diagram show in Figure 4.4 show the operation or the step of sending algorithms.

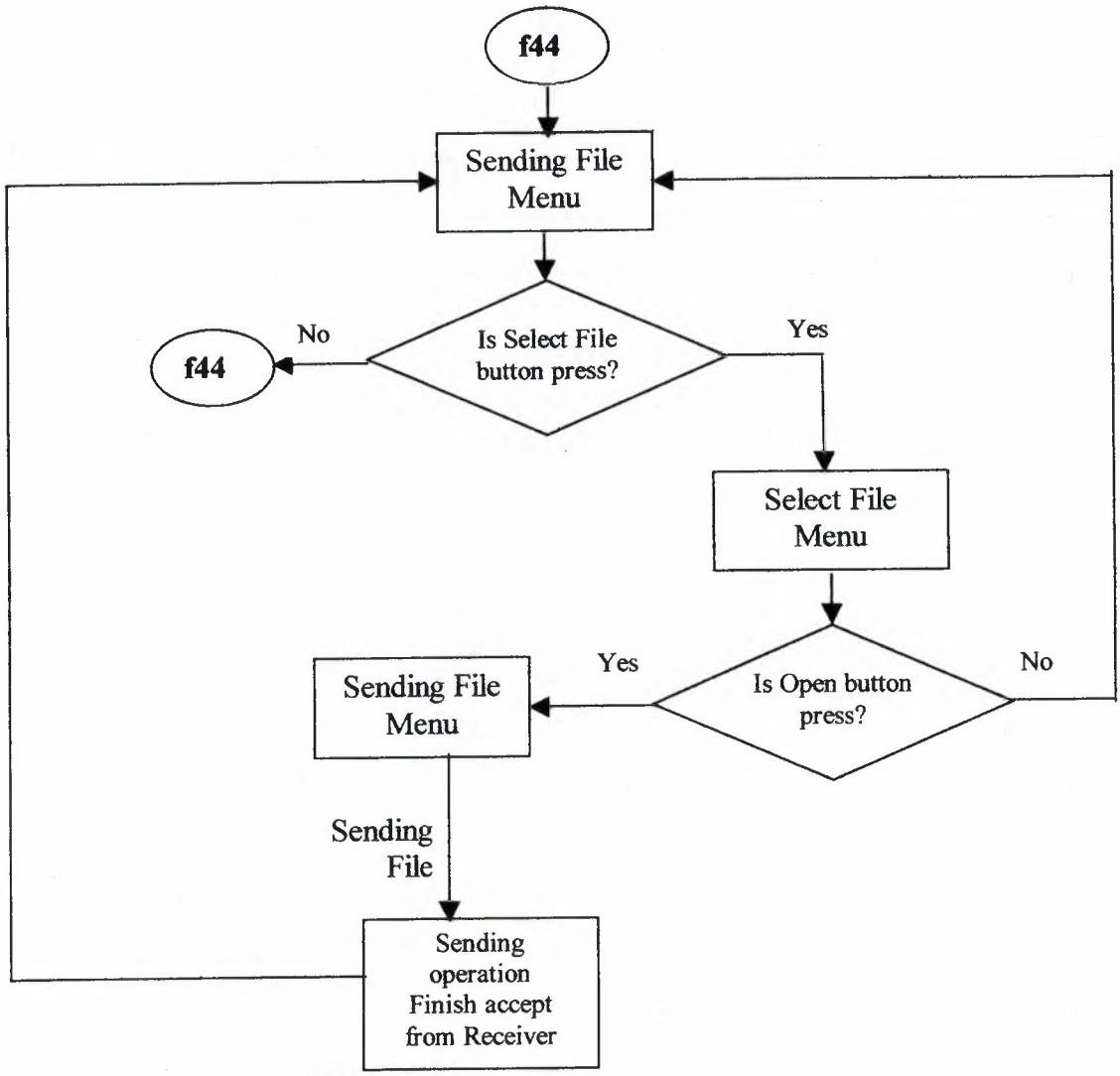

Figure 4.4 Sending algorithm

First thing you must select file to send it after that press the button send and wait the receiver to accept this file. If the receiver accepts this file the sending operation will be successful and return back if you want to send again. Or you can write directly type the name of file without selecting form selecting file menu. Note that if you press sending file button without selecting file, the program will be blocked. If you press cancel the program returns back to main menu.

4.3.4. Receiving algorithm

The below diagram show in Figure 4.5 show the operation or the step of sending algorithms.

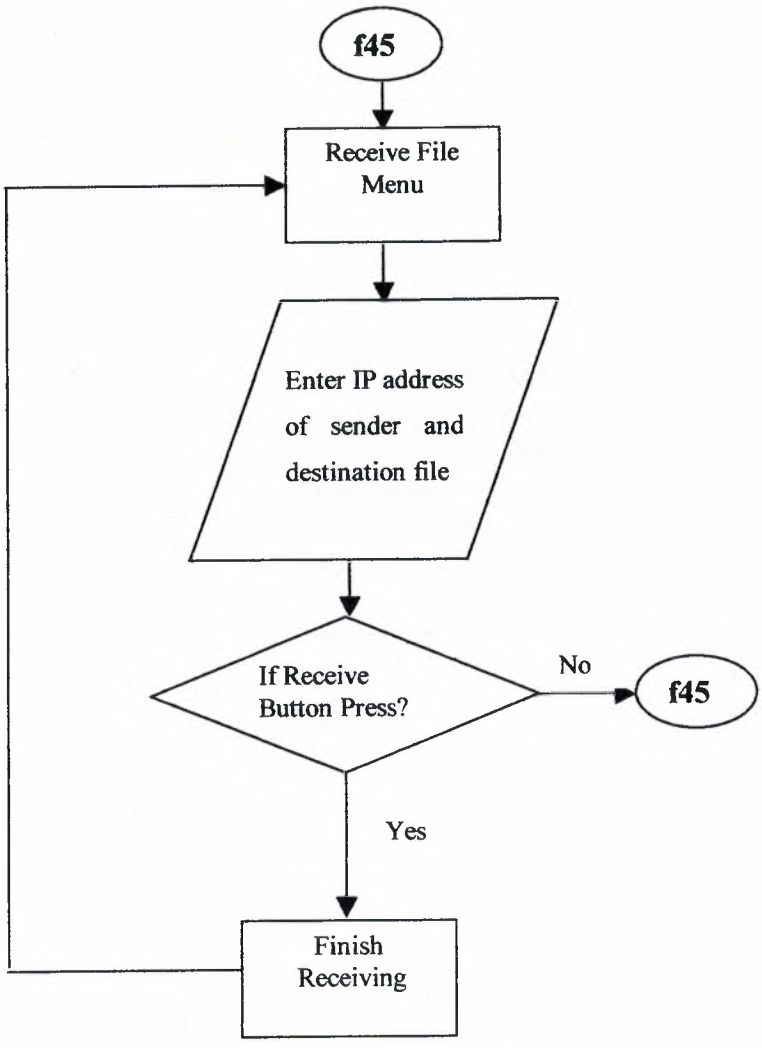

Figure 4.5 Receiving algorithm

First thing you must enter the IP address for sender and local destination that you will receive file in this location after that you will press button receive, you will receive file in that location if the sender IP is correct and he send that file, else the operation will be not successful. If you press return or refuse you will return to RC6 maim menu.

4.4. Using Application

4.4.1. Main Menu

The program start with the main menu interface as see in Figure 4.6 below that describes encryption, decryption, send, and receive file operation that easy interface for user to understand the program.

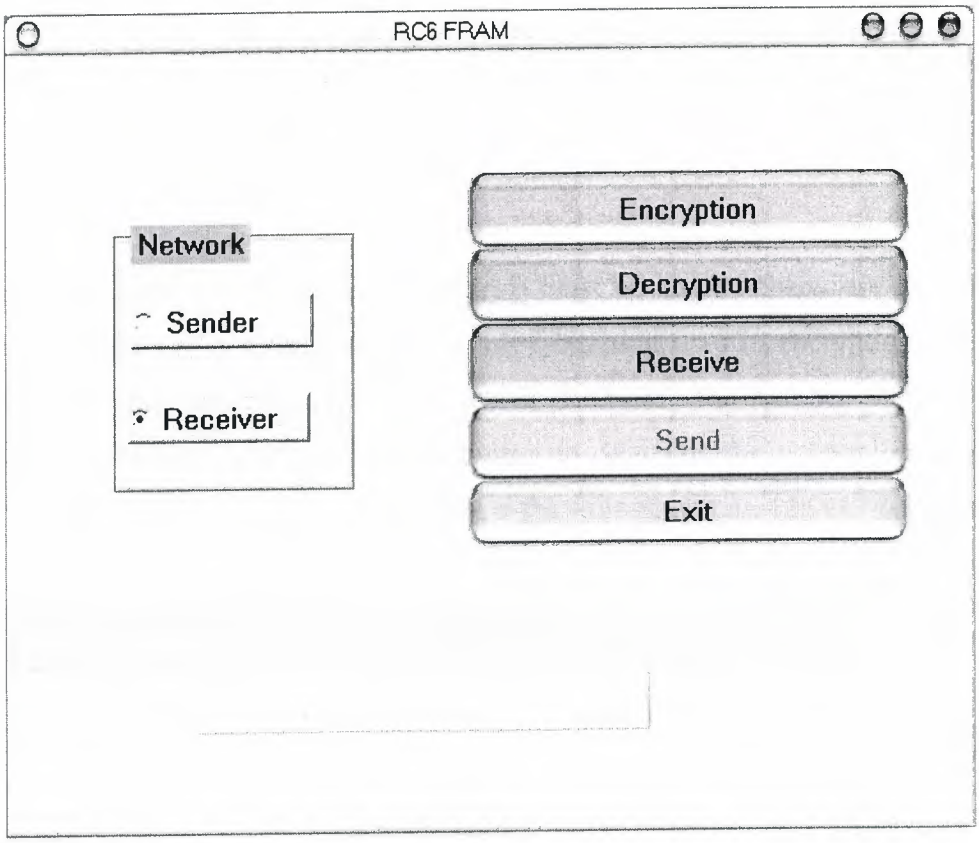

Figure 4.6 Main Menu

Also this Figure 4.6 contain network of sender and receiver. I mean by sender her the server, which publishes the IP address for the client (receiver). Now if the users want to encrypt, decrypt, send, and receive files using this application must follow below steps.

4.4.2. Encrypt File Menu

When user press encryption button the below Figure 4.7 will be appear that is interface for key schedule.

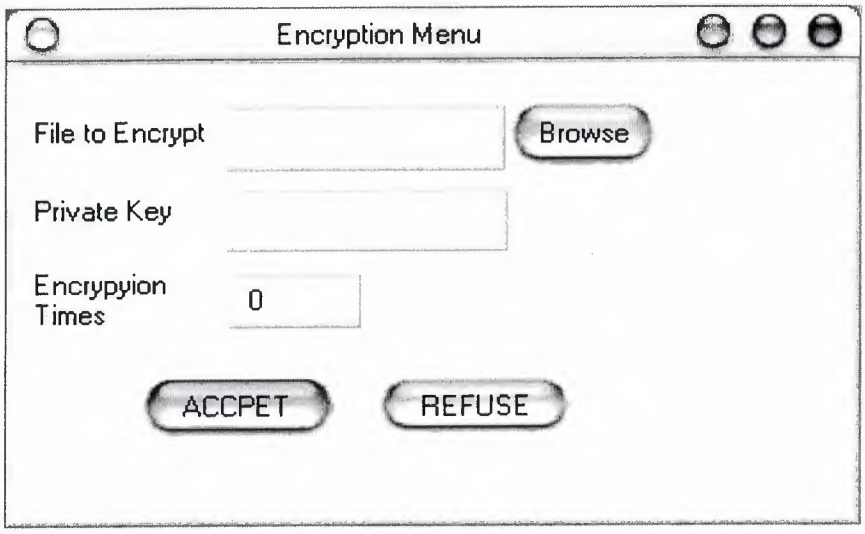

Figure 4.7 Key Schedule For Encryption

First, you must type the private key (PRKE) it must be sixteen characters (128-bits) to make key schedule to convert that key to 1408-bits that uses in encryption operation. After that type the second private key, which determine the number of encryptions times for each block. Then press accept button to complete the encryption operation. Now if the length of private key is not corrects the following massage will be appear.

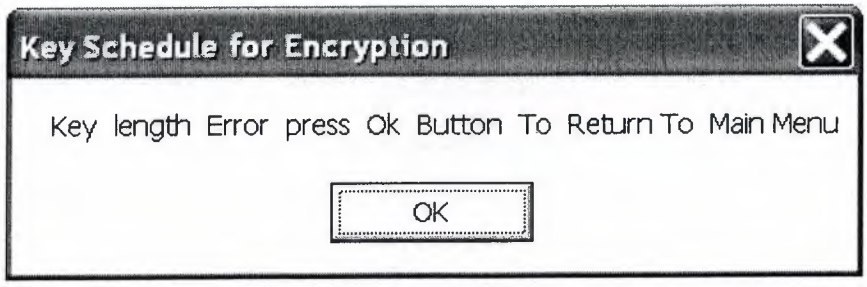

Figure 4.8 key Length Error

The Figure 4.8 indicates that when you press ok button the program return to main menu.

But before that you must choose file from browse button, after you press the below Figure 4.9 below will be appears to choose file to encrypt it.

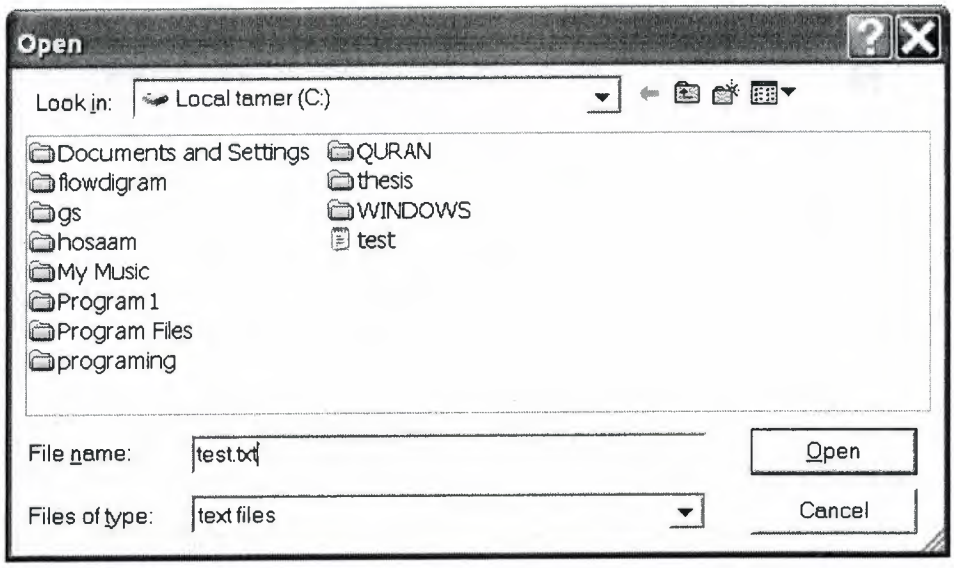

Figure 4.9 Encryption Selecting File Menu

When you select file to be encrypt press open button and press accept encryption operation for RC6 will be apply and the file will be encrypt and the fowling massage will be appear.

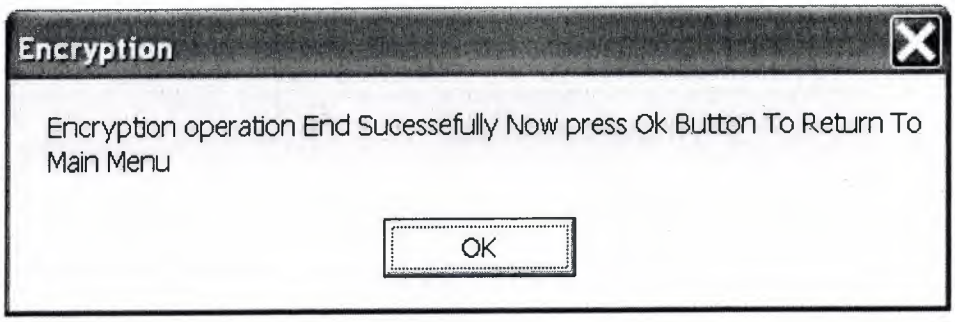

Figure 4.10 Successfully Encryption

The Figure 4.10 indicates that the encryption operation end successfully and the program will return to main menu. Note when the file encrypted, the extension of file will be converting to anther extension, which is (Cry1).

4.4.3. Decrypt File Menu

When user press decryption button the below Figure 4.7 will be appear that is interface for decryption file .

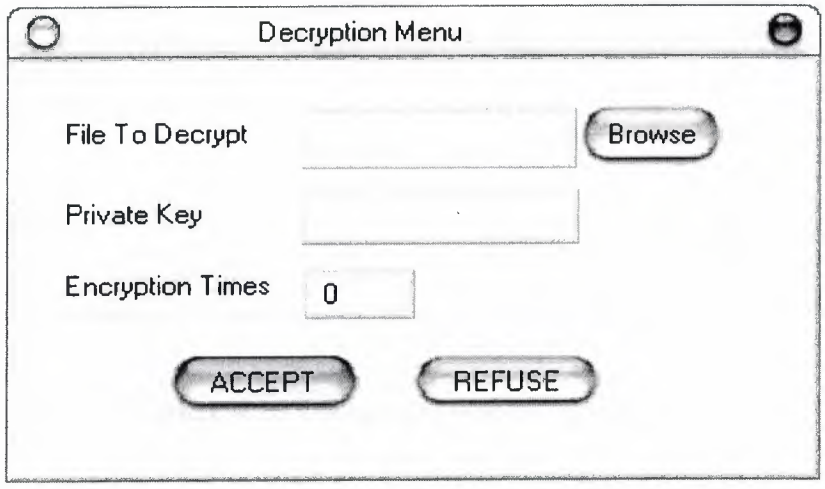

Figure 4.11 Key Schedule For Decryption

First, you must type the private key (PRKE) it must be sixteen characters (128-bits) to make key schedule to convert that key to 1408-bits that uses in decryption operation. After that type the second private key, which determine the number of Decryptions times for each block. Then press accept button to complete the decryption operation. Now if the length of private key is not corrects the following massage will be appear

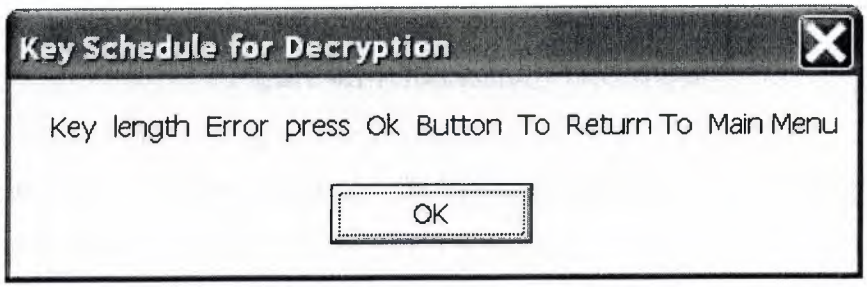

Figure 4.12 key Length Error

The Figure 4.12 indicates that when you press ok button the program return to main menu. But before that you must choose file from browse button, after you press the below Figure 4.9 will be appears to choose file to decrypt it. All the file appear with extension (Cry1)

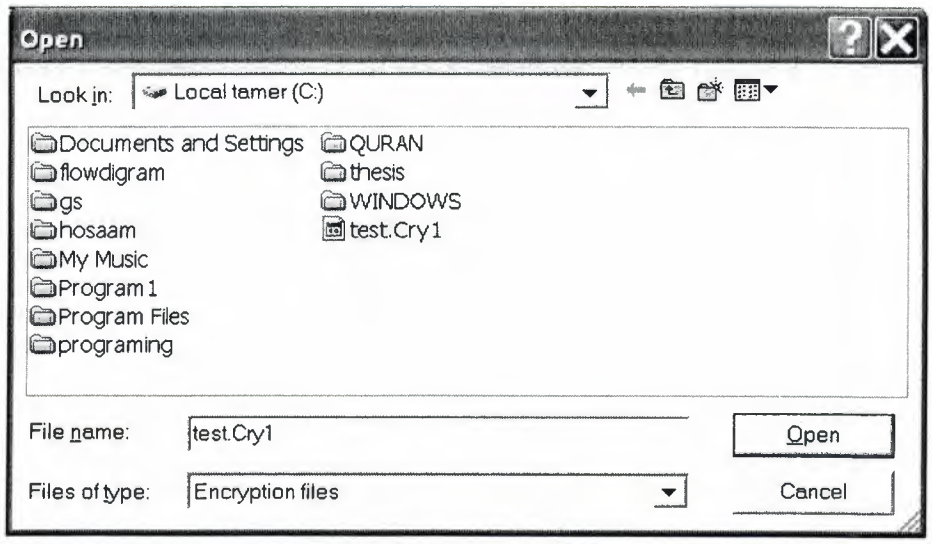

Figure 4.13 Decryption Select File Menu

When you selected file to be encrypt press open button the decryption algorithm for RC6 will be apply and the file will be decrypt and the fowling massage will be appear

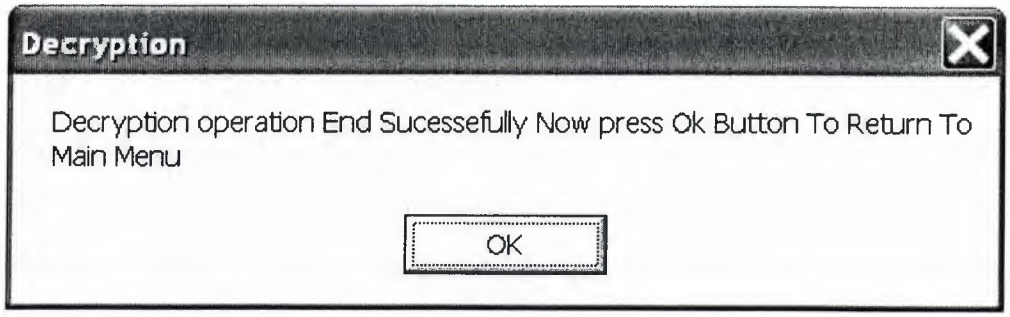

Figure 4.14 Successfully Decryption

The Figure 4.14 indicates that the decryption operation end successfully and the program will return to main menu.

Note: when the file decrypted, the extension of file will be converting from extension (Cry1) to original extension, which is (text).

4.4.4. Sending File Menu

When user press Send button from main menu the below Figure 4.15 will be appear to user and describe how to send file.

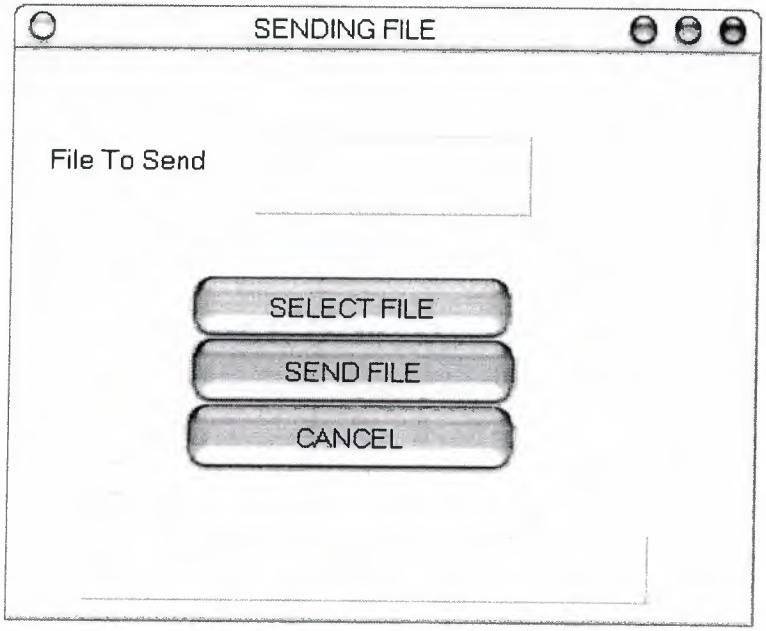

Figure 4.15 Sending File Menu

The Figure 4.15 indicate that the user must type the file name in edit box or select it from selecting menu as describe above. After that you can press send file to complete the sending operation and below massage will be appear which denote by Figure 4.16.

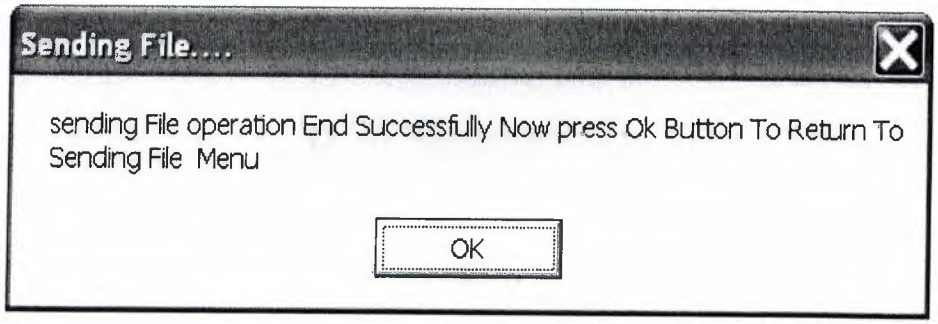

Figure 4.16 Sending Successfully

To satisfy successfully sending the receiver must connected at the same time with sender and accept the sending file.

But if you press the send file without type the name of the file or without selecting it. The below massage will be appear, which denoted by Figure 4.17.

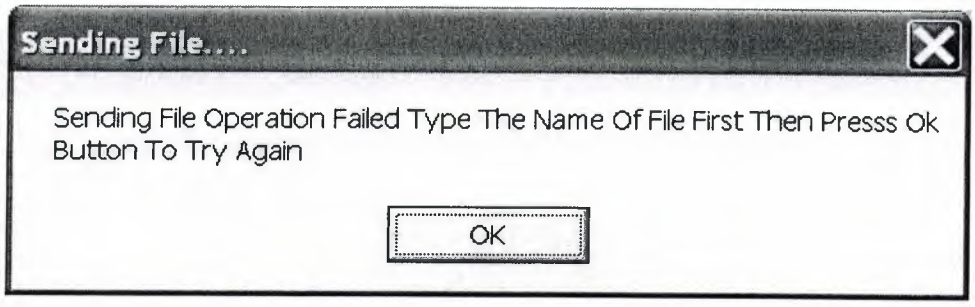

Figure 4.17 Sending Operation Failed

4.4.5. Receiving File Menu

When user press Receive button from main menu the below Figure 4.18 will be appear to user and describe how to receive file.

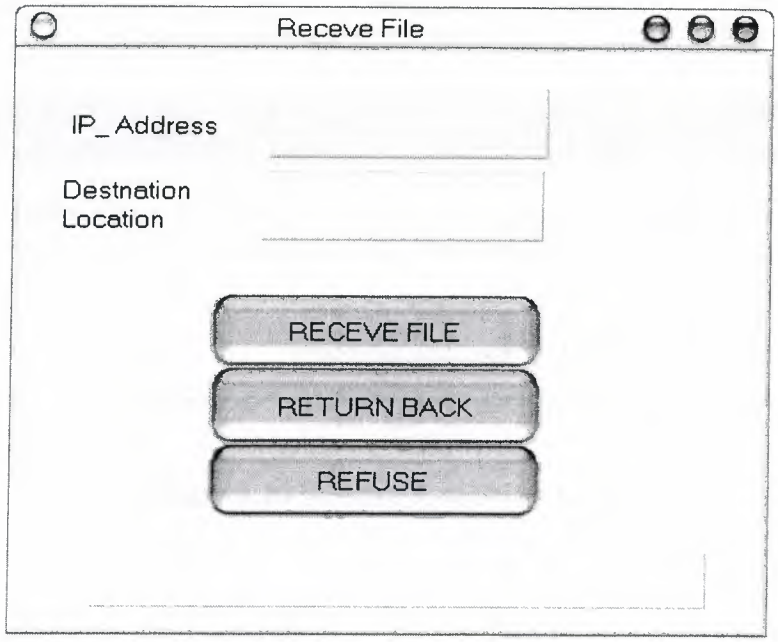

Figure 4.18 Receiving File Menu

The Figure 4.18 indicates that the user must type the IP address of sender and destination location for receiving file in edit box. After that you can press receive file to

complete the sending operation and below massage will be appear which denote by Figure 4.19.

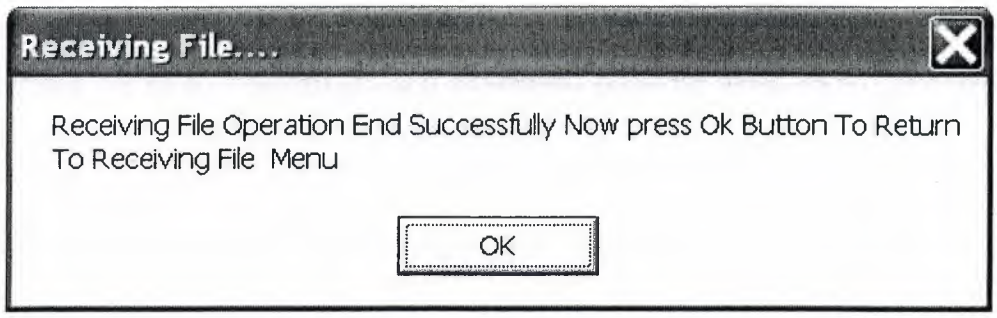

Figure 4.19 Sending Successfully

To satisfy successfully receiving, the sender must connect at the same time with receiver, and sender sent the file.

But if The IP address is wrong or destinations location didn't type in edit box the receiving operation is failed and appear the following massage

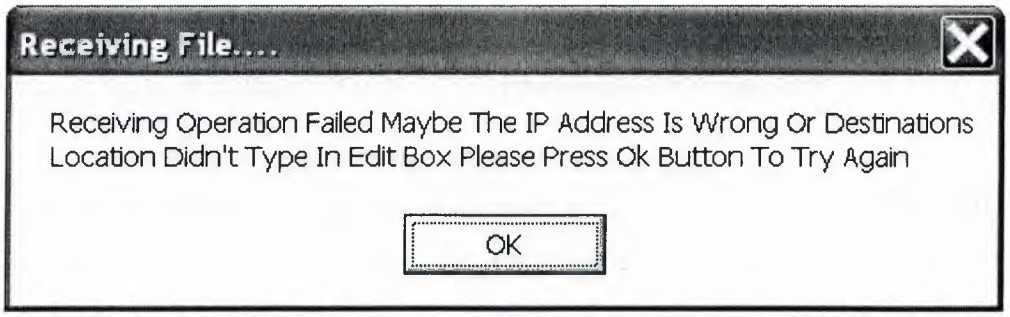

Figure 4.20 Sending Operation Failed

4.5. Summary

This chapter described how application works using RC6™, and how user deal with this application through flow charts and how they can encrypt, decrypt, send, receive files through networks.

CONCLUSION

In this thesis, it has given a detailed view on the security, cryptography algorithms especially analysis of the five candidates algorithms for AES, make analysis and modifications on RC6™ algorithm, and developed program using RC6™ through secure channel.

Next paragraph the important result obtained from this thesis:

- Information is becoming the most important resource of the economy and the society at large. One essential aspect for secure information and communications is the cryptography.

- There are several ways of classifying cryptographic algorithms. But best categorized based on key. There are two type of key-based algorithm: single key (symmetric key), asymmetric key.

- **Rijndeal.** It is a fast, flexible and elegant cipher. Rijndael is somewhat similar to SQUARE, and the lessons from SQUARE are incorporated in its design. The main worry about Rijndael is that it may not be conservative enough.

- **Serpent.** The main drawback of Serpent is that it is slower than the other finalists in software. On the other hand, it is very fast in hardware. The main selling point of Serpent is its very conservative number of rounds. Serpent does not have "fail-stop" mechanisms as in MARS, so in principle it is possible that a single major advance in cryptanalysis would yield a damaging attack. However, the large number of rounds makes such a possibility extremely remote.

- **Twofish.** This cipher was designed for flexibility, and indeed it offers a wide variety of implementation tradeoffs. It is also a very fast cipher. However, the same design for flexibility also resulted in a cipher, which is very hard to analyze. To obtain flexibility, the designers used many "tricks", whose security implications are not clear. The result is that among the five finalists.

- **RC6.** The main advantage of RC6 is its simplicity and speed. Its author, Ron Rivest, enjoys a well-deserved reputation in the cryptographic community, based on carefully crafted ciphers such as RSA, RC2, RC4, and RC5, which may serve as an indication for the suitability of the current design as well. The main argument against RC6 is "single point of failure" design.

- **MARS.** The main strength of MARS is its robustness. This was the main design goal, and MARS contains more "fail stop" mechanisms than any of the other finalists. Due to the heterogeneous structure and the large variety of "strong operations" in MARS, even a major advance in the cryptanalysis of any one of its components is very unlikely to lead to a significant attack against the overall cipher. MARS is also a very fast cipher in common use environments. The large number of fail-stop mechanisms in MARS makes its hardware implementation more involved than the other finalists, but it is still very small and cheap to implement in hardware, and is suitable to any real-life environment.

- Implementation of RC6™ using visual C++ 6, and apply it over network to protect data from attacker.

But we have observed that:

- It is fair to say that all five of the finalist algorithms very strong cipher algorithms. They are all symmetric ciphers, similar to AES in terms of block size and key lengths. It is my opinion that simplicity and speed are the most important selection criterion of the AES. From this perspective, author believe that RC6™ appears to be the best choice for the AES-FIPS because it has maximum points of its proven security, simplicity implementation, speed, its scaleable performance characteristics make it more flexible for the different demanding environments.

REFERENCES

[1] Willim Stallings. Cryptography and Network Security principles and practice. Third Edition. Prentice-Hall, Upper Saddle River, 2002.

[2] Wade Trappe, and Lawrence C. Washingtone. Cryptography Coding Theory. Prentice-Hall, Upper Saddle River, 2002.

[3] A. Menezes, P. van Oorschot, and S. Vanstone. Handbook of Applied Cryptography. CRC Press, 1996.

[4] SSH Communications Security. "Introduction to cryptography". Retrieved January 3 , 2004 from the Word Wide Web: "http://www.ssh.fi/support/cryptography/introduction/cryptanalysis.html".

[5] Xinmiao Zhang, and Kesham K. Parhi. Implementation Approaches for the Advanced Encryption Standared Algorithms. IEEE Circuits and Systems Magazine, Volume 2, Number 4, Fourth Quarter 2002.

[6] Şenol BektaŞ, Fakhraddin Mamedov, and Adnan Khashman. Graduate Studies: A Complete Reference. Near East University, Nicosia – 2001.

[7] George Coulouris, Jean Dollimore, and Time Kindberg, Distributed System concepts and Design. Third edition 2001.

[8] Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, and Niels Ferguson. "Twofish: A 128-Bit Block Cipher", 15 June 1998.

[9] Ronald L. Rivest, M.J.B. Robshaw, R. Sidney, and Y.L. Yin. "The RC6^{TM}Block Cipher". M.I.T. Laboratory for Computer Science, 545 Technology Square, Cambridge, RSA Laboratories, Version 1.1, August 20, 1998.

[10] Joan Daemen, and Vincent Rijmen. "A Specification for Rijndael, the AES Algorithm". version 3.2, 4th July 2001.

[11] Carolynn Burwick, Don Coppersmith, and Edward D'Avignon. "MARS -A Candidate cipher for AES". IBM Corporation, Revised, 22-September 1999.

[12] Ross Anderson[1], Eli Biham[2], and Lars Knudsen[3]. "Serpent: A Proposal for the Advanced Encryption Standard". [1] Cambridge University, England, [2] Technion, Haifa, Israel, [3] University of Bergen, Norway.

[13] DG Abraham, GM Dolan, GP Double, and JV Stevens. "Transaction Security System", IBM Systems Journal, v 30, no 2, 1991.

[14] Dhiren R. Pate. "The AES Winner". S V Regional Engineering College, Surat Gujarat, INDIA - 395 007.

[15] Kris Gaj, and Pawel Chodowiec. "Comparison of the hardware performance of the AES candidates using reconfigurable hardware". George Mason university.

[16] Ramesh Karri1, Kaijie Wu, Piyush Mishra1, and Yongkook Kim. "Concurrent Error Detection of Fault-Based Side-Channel Cryptanalysis of 128-Bit symmetric Block Ciphers". ECE Department, Polytechnic University.

[17] Security Technology Group Information Technology Laboratory NIST. "NIST's Efficiency Testing for Round1 AES Candidates".

[18] Ronald L. Rivest, and Yiqun Lisa Yin. "The Security of the RC6™ Block Cipher". RSA Laboratories, Version 1.0 - August 20, 1998.

[19] Sophia Antipolis. "Algorithm Selection for MAP Security". Ericsson, 28th-30th November, 2000.

[20] Reto Galli. "MARS encryption algorithm". ECE 575 Project - Winter 2000. Retrieved August 5, 2003 from the Word Wide Web: "http://www.islab.oregonstate.edu".

[21] Ross Anderson, and Eli Biham. "A Candidate Block Cipher for the Advanced Encryption Standard". Retrieved December 28, 2003 from the Word Wide Web: "http://www.nist.gov/aes".

[22] John Kelsey, Doug Whiting, and David Wagner. "Twofish - Performance vs. Other Block Ciphers", Counterpane Internet security, Inc., 2003. Retrieved October 6, 2003 from the Word Wide Web: "http://www.counterpane.com".

[23] Olli Cooper, "Cryptography", last updated 22 nd February 200. Retrieved October 18, 2003 from the Word Wide Web: "http://www.cs.bris.ac.uk/~cooper/Cryptography".

[24] Symeon Xenitellis. "guide to PKIs and Open–source Implementations". the Bikers-Home. Retrieved Decmber 22, 2003 from the Word Wide Web: "http://www.bikershome.com".

[25] Alan S H Lam. "Overview of Cryptographic Algorithms". Last Updated: Tuesday, 23-Nov-1999. Retrieved October 1, 2003 from the Word Wide Web: http://itec.erg.cuhk.edu.hk/archive/research/crypt1.htm.

[26] Simson Garfinkel, and Gene Spafford. Practical UNIX and Internet SecuritySecond Edition, April 1996.

[27] Mihir Bellare, and Phillip Rogaway. "Introduction to Modern Cryptography". Retrieved October 19, 2003 from the Word Wide Web: "http://www.cs.ucsd.edu/users/mihir/cse107/".

[28] Thomas Holenstein, Michèle Wigger, Renato Renner, and Robert König. "Information Security and cryptography Research Group". Institute of therotical Computer Science. Retrieved January 7, 2004 from the Word Wide "Web: http://www.crypto.ethz.ch/".

[29] Gary Kessler. "An Overview of Cryptography". Retrieved September 6, 2003 from the Word Wide Web: "http://www.garykessler.net/library".

[30] "Overview cryptography algorithm". Retrieved Augusts 22, 2003 from the Word Wide Web: "http://com2mac.postech.ac.kr/openschool".

[31] Carolyn Burwick[1], Don Coppersmith[2], Edward Avignon[1], and Mohammed Peyravian[3]. "The MARS Encryption Algorithm". [1]IBM Corporation, Poughkeepsie, NY 12601, USA, [2]IBM T. J. Watson Research, Yorktown Heights, NY 10598, USA, [3]IBM Corporation, Research Triangle Park, NC 27709, USA, August 27,1999.

[32] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, and N. Ferguson. "Twofish: A 128-bit Block Cipher". AES submission, June 1998, Retrieved November 23, 2003 from the Word Wide Web: "http://www.counterpane.com/twofish.html"

[33] Invor Horton, and Allan Stokes. "Beginning Visual C++ 6", First Edition, August 1998.

[34] PBS website. "The Playfair Cipher". Retrieved December 12, 2003 from the Word Wide web:"http://www.math.temple.edu/~renault/cryptology.html".

[35] softpedia™.com. "ActiveSkin4.3". Retrieved October 25, 2003 from the Word wide Web: "http://www.softpedia.com/public/scripts/downloadhero".

[36] John J. G. Savard. "A Cryptographic Compendium: The Hill Cipher". Retrieved November 23, 2003 from the Word Wide Web: "http://home.ecn.ab.ca/~jsavard/crypto/ro020103.htm"

[37] Peter Meyer. " An Introduction to the Use of Encryption ". Originally written January 994, Raised July 1997. Retrieved October 6, 2003 from the Word Wide Web: "http://www.hermetic.ch/index.html".

[38] William Stallings. Network Security Essentials: Applications And Standards. 2000 by Prentice-hall, Inc, Upper Saddle River, New Jersey 07458.

APPENDIX A

```
// RC6Dlg.cpp: implementation file
#include "stdafx.h"
#include "RC6.h"
#include "RC6Dlg.h"
#include "DlgRecever.h"
#include "DlgSending.h"
#include "math.h"
#include "KeyDlg.h"
#include "DecDlg.h"
#include <string.h>
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
#define  R 20
#define  W 32
CString Filename,encryfile;
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About
class CAboutDlg : public CDialog
{
public:
        CAboutDlg();
// Dialog Data
        //{{AFX_DATA(CAboutDlg)
        enum { IDD = IDD_ABOUTBOX };
        //}}AFX_DATA
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CAboutDlg)
```

```
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL
// Implementation
protected:
        //{{AFX_MSG(CAboutDlg)
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
        //{{AFX_DATA_INIT(CAboutDlg)
        //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CAboutDlg)
        //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
        //{{AFX_MSG_MAP(CAboutDlg)
        // No message handlers
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// CRC6Dlg dialog
CRC6Dlg::CRC6Dlg(CWnd* pParent /*=NULL*/)
        : CDialog(CRC6Dlg::IDD, pParent)
{
```

```
	//{{AFX_DATA_INIT(CRC6Dlg)
	m_Rad1 = -1;
	//}}AFX_DATA_INIT
	// Note that LoadIcon does not require a subsequent DestroyIcon in Win32
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CRC6Dlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CRC6Dlg)
	DDX_Radio(pDX, IDC_RADIO1, m_Rad1);
	//}}AFX_DATA_MAP
}
BEGIN_MESSAGE_MAP(CRC6Dlg, CDialog)
	//{{AFX_MSG_MAP(CRC6Dlg)
	ON_WM_SYSCOMMAND()
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
	ON_BN_CLICKED(IDC_BUTTON1, OnEnc)
	ON_BN_CLICKED(IDC_BUTTON2, OnDEC)
	ON_BN_CLICKED(IDC_BUTTON4, OnReceive)
	ON_BN_CLICKED(IDC_BUTTON5, OnSending)
	ON_BN_CLICKED(IDC_RADIO1, OnRadio1)
	ON_BN_CLICKED(IDC_RADIO2, OnRadio2)
	ON_BN_CLICKED(IDC_BUTTON3, OnButton3)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// CRC6Dlg message handlers
BOOL CRC6Dlg::OnInitDialog()
{
	CDialog::OnInitDialog();
```

```
// Add "About..." menu item to system menu.
// IDM_ABOUTBOX must be in the system command range.
ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
ASSERT(IDM_ABOUTBOX < 0xF000);
CMenu* pSysMenu = GetSystemMenu(FALSE);
if (pSysMenu != NULL)
{
        CString strAboutMenu;
        strAboutMenu.LoadString(IDS_ABOUTBOX);
        if (!strAboutMenu.IsEmpty())
        {
                pSysMenu->AppendMenu(MF_SEPARATOR);
                pSysMenu->AppendMenu(MF_STRING,IDM_ABOUTBOX,
                trAboutMenu);
        }
}
// Set the icon for this dialog.  The framework does this automatically
//  when the application's main window is not a dialog
SetIcon(m_hIcon, TRUE);            // Set big icon
SetIcon(m_hIcon, FALSE);           // Set small icon
// TODO: Add extra initialization here0
m_Rad1=1;
GetDlgItem ( IDC_BUTTON5) -> EnableWindow ( FALSE ) ;
UpdateData(FALSE);
CComQIPtr<ISkin> pSkin = GetDlgItem(IDC_SKIN22)-
>GetControlUnknown();
pSkin->ApplySkin((long)m_hWnd);
return TRUE;  // return TRUE  unless you set the focus to a control
}

void CRC6Dlg::OnSysCommand(UINT nID, LPARAM lParam)
{
        if ((nID & 0xFFF0) == IDM_ABOUTBOX)
```

```
        {
                CAboutDlg dlgAbout;
                dlgAbout.DoModal();
        }
        else
        {
                CDialog::OnSysCommand(nID, lParam);
        }
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CRC6Dlg::OnPaint()
{
        if (IsIconic())
        {
                CPaintDC dc(this); // device context for painting
                SendMessage(WM_ICONERASEBKGND, (WPARAM)
                dc.GetSafeHdc(), 0);
                // Center icon in client rectangle
                int cxIcon = GetSystemMetrics(SM_CXICON);
                int cyIcon = GetSystemMetrics(SM_CYICON);
                CRect rect;
                GetClientRect(&rect);
                int x = (rect.Width() - cxIcon + 1) / 2;
                int y = (rect.Height() - cyIcon + 1) / 2;
                // Draw the icon
                dc.DrawIcon(x, y, m_hIcon);
        }
        else
        {
```

```
            CDialog::OnPaint();
        }
}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CRC6Dlg::OnQueryDragIcon()
{
        return (HCURSOR) m_hIcon;
}

void CRC6Dlg::OnEnc()
{
        // TODO: Add your control notification handler code here
        UpdateData(TRUE);
        CKeyDlg dlg;
        int Key_leng;
        char getkey[16];
        CProgressCtrl *p=(CProgressCtrl*) GetDlgItem(IDC_PROGRESS1);
        p->SetRange(0,1000);
        p->SetPos(0);
        if(dlg.DoModal()!=IDCANCEL)
        {
                Key_leng=dlg.m_Key.GetLength();
                if (Key_leng==16)//lenght of key is 128 or 16 bayte
                {
                        strcpy(getkey,dlg.m_Key);
                        KeyGen(getkey);
                        UpdateData(TRUE);
                        Filename= dlg.m_encfile;
                        encryfile= dlg.decfile;
                        int time=dlg.m_iteration;
                        Encryption(time);
```

```
                    for(int j=1;j<1000;j++)
                    p->SetPos(j);
                    MessageBox("Encryption operation
                    Sucessefully","Encryption",MB_OK);
                    p->SetPos(0);
                    }else
                    MessageBox("Encryption Operation Failed Enter Correct Private
                    Key","Encryption",MB_OK);
        }

}

void CRC6Dlg::OnDEC()
{
        // TODO: Add your control notification handler code here
        UpdateData(TRUE);
        CDecDlg dlg;
        int Key_leng;
        char getkey[16];
        CProgressCtrl *p=(CProgressCtrl*) GetDlgItem(IDC_PROGRESS1);
        p->SetRange(0,1000);
        p->SetPos(0);
        if(dlg.DoModal()!=IDCANCEL)
        {
                Key_leng=dlg.m_pkey.GetLength();
                        if (Key_leng==16)//lenght of key is 128 or 16 bayte
                        {
                        strcpy(getkey,dlg.m_pkey);
                        KeyGen(getkey);
                        UpdateData(TRUE);
                        encryfile= dlg.m_encfile;
                        Filename= dlg.decfile;
                        int time=dlg.m_iteration;
```

```
                    Decryption(time);
                    for(int j=1;j<1000;j++)
                    p->SetPos(j);
                    MessageBox("Decryption Sucessefully","Decryption",MB_OK);
                     p->SetPos(0);
                    }else MessageBox("Decryption Operation Failed Enter
                    Correct Private Key","Decryption",MB_OK);
        }
}

DWORD CRC6Dlg::RightRotate(DWORD dwVar, DWORD dwOffset)
{
        DWORD temp1, temp2;
        temp1 = dwVar << (W - dwOffset);
        temp2 = dwVar >> dwOffset;
        temp2 = temp2 | temp1;
        return temp2;
}

DWORD CRC6Dlg::OffsetAmount(DWORD dwVar)
{
        int nLgw = (int)(log((double)W)/log(2.0));
        dwVar = dwVar << (W - nLgw);
        dwVar = dwVar >> (W - nLgw);
        return dwVar;
}

DWORD CRC6Dlg::LeftRotate(DWORD dwVar, DWORD dwOffset)
{
        DWORD temp1, temp2;
        temp1 = dwVar >> (W - dwOffset);
        temp2 = dwVar << dwOffset;
        temp2 = temp2 | temp1;
```

```
        return temp2;
}

void CRC6Dlg::KeyGen(char dwKey1[16])
{
        DWORD P32 = 0xB7E15163;
        DWORD Q32 = 0x9E3779B9;
        DWORD i, A, B,k,j,t;
        DWORD  dwKey[4];
        for(k= 0; k <4; ++k)
        dwKey[k] = ((DWORD *)dwKey1)[k];
        m_dwS[0] = P32;
        for(i = 1; i < 2 * R + 4; i++){
                m_dwS[i] = m_dwS[i - 1] + Q32;
                i = A = B =j= 0;
                t=3;
                int v = 3 * max(1, 2 * R + 4);
                for(int s = 1; s <= v; s++)
                {
                A = m_dwS[i] = LeftRotate(m_dwS[i] + A + B, OffsetAmount(3));
                B = dwKey[j] = LeftRotate(dwKey[j] + A + B, OffsetAmount(A + B));
                i = (i + 1) % (2 * R + 4);
                j = (j == t ? 0 : j + 1);
        }

}

void CRC6Dlg::Encryption(int time1)
{
        CFile MyFile,MyFile1;
        DWORD pdwTemp[4];
        MyFile.Open(Filename, CFile::modeReadWrite);
        MyFile1.Open(encryfile,CFile::modeCreate | CFile::modeWrite );
```

```
        DWORD Len;
        Len=MyFile.GetLength();
        DWORD DwLen=Len/16;
        DWORD DwRem=Len%16;
        for(int i=0;i<DwLen;i++){
        MyFile.Read(pdwTemp,16);//the second parameter is the No of the
        for(int i=0;i<time1;i++){
                pdwTemp[0] = (pdwTemp[0] - m_dwS[2 * R + 2]);
                pdwTemp[2] = (pdwTemp[2] - m_dwS[2 * R + 3]);
                for(int  j = R; j >= 1; j--)
        {
        DWORD temp = pdwTemp[3];
        pdwTemp[3] = pdwTemp[2];
        pdwTemp[2] = pdwTemp[1];
        pdwTemp[1] = pdwTemp[0];
        pdwTemp[0] = temp;
        DWORD t = LeftRotate((pdwTemp[1] * (2 * pdwTemp[1] + 1)),
        OffsetAmount((DWORD)(log((double)W)/log(2.0))));
        DWORD u = LeftRotate((pdwTemp[3] * (2 * pdwTemp[3] + 1)),
        OffsetAmount((DWORD)(log((double)W)/log(2.0))));
        pdwTemp[0] = (RightRotate((pdwTemp[0] - m_dwS[2 * j]),
        OffsetAmount(u))) ^ t;
        pdwTemp[2] = (RightRotate((pdwTemp[2] - m_dwS[2 * j + 1]),
        OffsetAmount(t))) ^ u;
        }
        pdwTemp[1] = (pdwTemp[1] - m_dwS[0]);
        pdwTemp[3] = (pdwTemp[3] - m_dwS[1]);
        }
        MyFile1.Write(pdwTemp,16);
}

/// for reminder begin
        if (DwRem!=0){
```

```
        pdwTemp[0]=0x00000000;
        pdwTemp[1]=0x00000000;
        pdwTemp[2]=0x00000000;
        pdwTemp[3]=0x00000000;
        MyFile.Read(pdwTemp,16);//the second parameter is the No of  key
        for(int i=0;i<time1;i++){
        pdwTemp[0] = (pdwTemp[0] - m_dwS[2 * R + 2]);
        pdwTemp[2] = (pdwTemp[2] - m_dwS[2 * R + 3]);
        for(int  j = R; j >= 1; j--)
        {
                DWORD temp = pdwTemp[3];
                pdwTemp[3] = pdwTemp[2];
                pdwTemp[2] = pdwTemp[1];
                pdwTemp[1] = pdwTemp[0];
                pdwTemp[0] = temp;
                DWORD t = LeftRotate((pdwTemp[1] * (2 * pdwTemp[1] + 1)),
                OffsetAmount((DWORD)(log((double)W)/log(2.0))));
                DWORD u = LeftRotate((pdwTemp[3] * (2 * pdwTemp[3] + 1)),
                OffsetAmount((DWORD)(log((double)W)/log(2.0))));
                pdwTemp[0] = (RightRotate((pdwTemp[0] - m_dwS[2 * j]),
                OffsetAmount(u))) ^ t;
                pdwTemp[2] = (RightRotate((pdwTemp[2] - m_dwS[2 * j + 1]),
                OffsetAmount(t))) ^ u;
        }
        pdwTemp[1] = (pdwTemp[1] - m_dwS[0]);
        pdwTemp[3] = (pdwTemp[3] - m_dwS[1]);
        }
        MyFile1.Write(pdwTemp,16);
}//end
MyFile1.Close();
MyFile.Close();
MyFile.Remove(Filename);
}
```

```
void CRC6Dlg::Decryption(int time1)
{
        CFile MyFile,MyFile2;
        DWORD pdwTemp[4];
        MyFile.Open(Filename, CFile::modeCreate | CFile::modeWrite );
        MyFile2.Open(encryfile, CFile::modeRead);
        DWORD Len=MyFile2.GetLength();
        DWORD DwLen=Len/16;
        DWORD DwRem=Len%16;
        for (int i=0;i<DwLen;i++){
        MyFile2.Read(pdwTemp,16);//the second p
        for(int i=0;i<time1;i++){
        pdwTemp[1] = (pdwTemp[1] + m_dwS[0]);
        pdwTemp[3] = (pdwTemp[3] + m_dwS[1]);
        for( int j = 1; j <= R; j++)
        {
                DWORD t = LeftRotate((pdwTemp[1] * (2 * pdwTemp[1] + 1)),
                OffsetAmount((DWORD)(log((double)W)/log(2.0))));
                DWORD u = LeftRotate((pdwTemp[3] * (2 * pdwTemp[3] + 1)),
                OffsetAmount((DWORD)(log((double)W)/log(2.0))));
                pdwTemp[0] = (LeftRotate(pdwTemp[0] ^ t, OffsetAmount(u)) +
                m_dwS[2 * j]);
                pdwTemp[2] = (LeftRotate(pdwTemp[2] ^ u, OffsetAmount(t)) +
                m_dwS[2 *  j + 1]);
                DWORD temp = pdwTemp[0];
                pdwTemp[0] = pdwTemp[1];
                pdwTemp[1] = pdwTemp[2];
                pdwTemp[2] = pdwTemp[3];
                pdwTemp[3] = temp;}
                pdwTemp[0] = (pdwTemp[0] + m_dwS[2 * R + 2]);
                pdwTemp[2] = (pdwTemp[2] + m_dwS[2 * R + 3]);
        }
                MyFile.Write(pdwTemp,16);
```

```
}
/// for reminder begin
if (DwRem!=0){
pdwTemp[0]=0x00000000;
pdwTemp[1]=0x00000000;
pdwTemp[2]=0x00000000;
pdwTemp[3]=0x00000000;
MyFile2.Read(pdwTemp,16);
for(int i=0;i<time1;i++){
pdwTemp[1] = (pdwTemp[1] + m_dwS[0]);
pdwTemp[3] = (pdwTemp[3] + m_dwS[1]);
for( int j = 1; j <= R; j++)
{
        DWORD t = LeftRotate((pdwTemp[1] * (2 * pdwTemp[1] + 1)),
        OffsetAmount((DWORD)(log((double)W)/log(2.0))));
        DWORD u = LeftRotate((pdwTemp[3] * (2 * pdwTemp[3] + 1)),
        OffsetAmount((DWORD)(log((double)W)/log(2.0))));
        pdwTemp[0] = (LeftRotate(pdwTemp[0] ^ t, OffsetAmount(u)) +
        m_dwS[2 * j]);
        pdwTemp[2] = (LeftRotate(pdwTemp[2] ^ u, OffsetAmount(t)) +
        m_dwS[2 * j + 1]);
        DWORD temp = pdwTemp[0];
        pdwTemp[0] = pdwTemp[1];
        pdwTemp[1] = pdwTemp[2];
        pdwTemp[2] = pdwTemp[3];
        pdwTemp[3] = temp;
        }
        pdwTemp[0] = (pdwTemp[0] + m_dwS[2 * R + 2]);
        pdwTemp[2] = (pdwTemp[2] + m_dwS[2 * R + 3]);

}
MyFile.Write(pdwTemp,16);
```

```
        }
        MyFile.Close();
        MyFile2.Close();
        MyFile2.Remove(encryfile);
        ///end
}

void CRC6Dlg::OnReceive()
{
        // TODO: Add your control notification handler code here
        CDlgRecever dlg1;
        dlg1.DoModal();
}

void CRC6Dlg::OnSending()
{
        CDlgSending dlg;
        dlg.DoModal();
}

void CRC6Dlg::OnRadio1()
{
        // TODO: Add your control notification handler code here
        UpdateData ( TRUE ) ;
        if(m_Rad1==0)
        {
                GetDlgItem ( IDC_BUTTON5 ) -> EnableWindow ( TRUE ) ;
                GetDlgItem ( IDC_BUTTON4 ) -> EnableWindow ( FALSE ) ;
        }
        else{
                GetDlgItem ( IDC_BUTTON5 ) -> EnableWindow ( FALSE ) ;
                GetDlgItem ( IDC_BUTTON4 ) -> EnableWindow ( TRUE ) ;
        }
```

```
}

void CRC6Dlg::OnRadio2()
{
        // TODO: Add your control notification handler code here
        UpdateData ( TRUE ) ;
        if(m_Rad1==1){
        GetDlgItem ( IDC_BUTTON4 ) -> EnableWindow ( TRUE ) ;
        GetDlgItem ( IDC_BUTTON5 ) -> EnableWindow ( FALSE ) ;
        }
        else{
        GetDlgItem ( IDC_BUTTON5 ) -> EnableWindow ( FALSE ) ;
        GetDlgItem ( IDC_BUTTON4 ) -> EnableWindow ( TRUE ) ;
        }
}

void CRC6Dlg::OnButton3()
{
        CDialog::OnCancel();
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// RC6Dlg.h : header file
#if
!defined(AFX_RC6DLG_H__0F7B45E7_E761_11D7_9DDC_8BC8A7F56F3F__INC
LUDED_)
#define
AFX_RC6DLG_H__0F7B45E7_E761_11D7_9DDC_8BC8A7F56F3F__INCLUDED_
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
#define R 20
#import "actskn43.ocx" no_implementation raw_interfaces_only raw_native_types
```

```
using namespace ACTIVESKINLib;
#include "atlbase.h"
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// CRC6Dlg dialog
class CRC6Dlg : public CDialog
{
// Construction
public:
	CRC6Dlg(CWnd* pParent = NULL);		// standard constructor
// Dialog Data
	//{{AFX_DATA(CRC6Dlg)
	enum { IDD = IDD_RC6_DIALOG };
	int	m_Rad1;
	//}}AFX_DATA
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CRC6Dlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);	// DDX/DDV
support
	//}}AFX_VIRTUAL
// Implementation
protected:
	DWORD m_dwS[2*R+4];
	HICON m_hIcon;
	// Generated message map functions
	//{{AFX_MSG(CRC6Dlg)
	virtual BOOL OnInitDialog();
	afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
	afx_msg void OnPaint();
	afx_msg HCURSOR OnQueryDragIcon();
	afx_msg void OnEnc();
	afx_msg void OnDEC();
	afx_msg void OnReceive();
```

```
        afx_msg void OnSending();
        afx_msg void OnRadio1();
        afx_msg void OnRadio2();
        afx_msg void OnButton3();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
private:
        void Encryption(int time1);
        void Decryption(int time1);
        void KeyGen(char  dwKey1[16]);
        DWORD LeftRotate(DWORD dwVar, DWORD dwOffset);
        DWORD OffsetAmount(DWORD dwVar);
        DWORD RightRotate(DWORD dwVar, DWORD dwOffset);
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the
previous line.
#endif //
!defined(AFX_RC6DLG_H__0F7B45E7_E761_11D7_9DDC_8BC8A7F56F3F__INC
LUDED_)
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// RC6.cpp : Defines the class behaviors for the application.
#include "stdafx.h"
#include "RC6.h"
#include "RC6Dlg.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
// CRC6App
BEGIN_MESSAGE_MAP(CRC6App, CWinApp)
//{{AFX_MSG_MAP(CRC6App)
// NOTE - the ClassWizard will add and remove mapping macros here.
//    DO NOT EDIT what you see in these blocks of generated code!
//}}AFX_MSG
ON_COMMAND(ID_HELP, CWinApp::OnHelp)
END_MESSAGE_MAP()
/////////////////////////////////////////////////////////////////////////////
// CRC6App construction
CRC6App::CRC6App()
{
        // TODO: add construction code here,
        // Place all significant initialization in InitInstance
}
/////////////////////////////////////////////////////////////////////////////
// The one and only CRC6App object
CRC6App theApp;
// CRC6App initialization
BOOL CRC6App::InitInstance()
{
        if (!AfxSocketInit())
        {
                AfxMessageBox(IDP_SOCKETS_INIT_FAILED);
                return FALSE;
        }

AfxEnableControlContainer();
// Standard initialization
// If you are not using these features and wish to reduce the size
//  of your final executable, you should remove from the following
//  the specific initialization routines you do not need.
```

```
#ifdef _AFXDLL
        Enable3dControls();                    // Call this when using MFC in a shared
DLL
#else
        Enable3dControlsStatic();      // Call this when linking to MFC statically
#endif

CRC6Dlg dlg;
m_pMainWnd = &dlg;
int nResponse = dlg.DoModal();
if (nResponse == IDOK)
{
        // TODO: Place code here to handle when the dialog is
        //  dismissed with OK
}
else if (nResponse == IDCANCEL)
{
        // TODO: Place code here to handle when the dialog is
        //  dismissed with Cancel
}
        // Since the dialog has been closed, return FALSE so that we exit the
        //  application, rather than start the application's message pump.
        return FALSE;
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// RC6.h : main header file for the RC6 application
#if
!defined(AFX_RC6_H__0F7B45E5_E761_11D7_9DDC_8BC8A7F56F3F__INCLUD
ED_)
#define
AFX_RC6_H__0F7B45E5_E761_11D7_9DDC_8BC8A7F56F3F__INCLUDED_
#if _MSC_VER > 1000
#pragma once
```

```
#endif // _MSC_VER > 1000
#ifndef __AFXWIN_H__
#error include 'stdafx.h' before including this file for PCH
#endif
#include "resource.h"           // main symbols

/////////////////////////////////////////////////////////////////////////////
// CRC6App:
// See RC6.cpp for the implementation of this class
//
class CRC6App : public CWinApp
{
public:
        CRC6App();
// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CRC6App)
        public:
        virtual BOOL InitInstance();
        //}}AFX_VIRTUAL

// Implementation

        //{{AFX_MSG(CRC6App)
        // NOTE - the ClassWizard will add and remove member functions here.
        //    DO NOT EDIT what you see in these blocks of generated code !
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
/////////////////////////////////////////////////////////////////////////////
//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the
previous line.
```

```
#endif //
!defined(AFX_RC6_H__0F7B45E5_E761_11D7_9DDC_8BC8A7F56F3F__INCLUD
ED_)
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// KeyDlg.cpp : implementation file
#include "stdafx.h"
#include "RC6.h"
#include "KeyDlg.h"
#include "RC6Dlg.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// CKeyDlg dialog
CKeyDlg::CKeyDlg(CWnd* pParent /*=NULL*/)
        : CDialog(CKeyDlg::IDD, pParent)
{
        //{{AFX_DATA_INIT(CKeyDlg)
        m_Key = _T("");
        m_iteration = 0;
        m_encfile = _T("");
        //}}AFX_DATA_INIT
}

void CKeyDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CKeyDlg)
        DDX_Text(pDX, IDC_EDIT1,m_Key);
        DDX_Text(pDX, IDC_EDIT2,m_iteration);
```

```
    DDX_Text(pDX, IDC_EDIT3, m_encfile);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CKeyDlg, CDialog)
    //{{AFX_MSG_MAP(CKeyDlg)
    ON_BN_CLICKED(IDC_BUTTON1, OnButton1)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// CKeyDlg message handlers
void CKeyDlg::OnOK()
{
    // TODO: Add extra validation here
    CDialog::OnOK();
}

void CKeyDlg::OnCancel()
{
    // TODO: Add extra cleanup here
    CDialog::OnCancel();
}

BOOL CKeyDlg::OnInitDialog()
{
    CDialog::OnInitDialog();
    // TODO: Add extra initialization here
    CComQIPtr<ISkin> pSkin = GetDlgItem(IDC_SKIN22)-
    GetControlUnknown();
    pSkin->ApplySkin((long)m_hWnd);
    return TRUE;  // return TRUE unless you set the focus to a control
    // EXCEPTION: OCX Property Pages should return FALSE
}
```

```
void CKeyDlg::OnButton1()
{
        // TODO: Add your control notification handler code here
        //CString encfile;
        CFileDialog m_IdFile (TRUE,NULL, NULL, OFN_HIDEREADONLY,"text
        iles|*.txt");
        m_IdFile.m_ofn.lpstrInitialDir = "c:\\";
        if (m_IdFile.DoModal() != IDCANCEL ){
        m_encfile = m_IdFile.GetPathName();
        decfile=(m_IdFile.GetFileTitle() + ".Cry1");
        UpdateData(FALSE);
        }else MessageBox("Please Select File To Complet
        ncryption","Encryption",MB_OK);
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
if
!defined(AFX_KEYDLG_H__0F7B45EF_E761_11D7_9DDC_8BC8A7F56F3F__INC
LUDED_)
#define
AFX_KEYDLG_H__0F7B45EF_E761_11D7_9DDC_8BC8A7F56F3F__INCLUDED
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// KeyDlg.h : header file
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// CKeyDlg dialog
class CKeyDlg : public CDialog
{
// Construction
public:
        CString decfile;
        CKeyDlg(CWnd* pParent = NULL);   // standard constructor
```

```
// Dialog Data
        //{{AFX_DATA(CKeyDlg)
        enum { IDD = IDD_KEY_DIALOG };
        CString         m_Key;
        int             m_iteration;
        CString         m_encfile;
        //}}AFX_DATA
// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CKeyDlg)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL
// Implementation
protected:
        // Generated message map functions
        //{{AFX_MSG(CKeyDlg)
        virtual void OnOK();
        virtual void OnCancel();
        virtual BOOL OnInitDialog();
        afx_msg void OnButton1();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the
previous line.
#endif //
!defined(AFX_KEYDLG_H__0F7B45EF_E761_11D7_9DDC_8BC8A7F56F3F__INC
LUDED_).
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
// stdafx.h : include file for standard system include files,
//  or project specific include files that are used frequently, but
//      are changed infrequently
//

#if
!defined(AFX_STDAFX_H__0F7B45E9_E761_11D7_9DDC_8BC8A7F56F3F__INC
LUDED_)
#define
AFX_STDAFX_H__0F7B45E9_E761_11D7_9DDC_8BC8A7F56F3F__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#define VC_EXTRALEAN    // Exclude rarely-used stuff from Windows headers
#include <afxwin.h>         // MFC core and standard components
#include <afxext.h>         // MFC extensions
#include <afxdisp.h>        // MFC Automation classes
#include <afxdtctl.h> // MFC support for Internet Explorer 4 Common Controls
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>                     // MFC support for Windows Common Controls
#endif // _AFX_NO_AFXCMN_SUPPORT

#include <afxsock.h>            // MFC socket extensions

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the
previous line.

#endif //
!defined(AFX_STDAFX_H__0F7B45E9_E761_11D7_9DDC_8BC8A7F56F3F__INC
LUDED_)
```

```
// DlgSending.cpp : implementation file
#include "stdafx.h"
#include "RC6.h"
#include "DlgSending.h"
#include "RC6Dlg.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
/////////////////////////////////////////////////////////////////////////////
// CDlgSending dialog
CDlgSending::CDlgSending(CWnd* pParent /*=NULL*/)
        : CDialog(CDlgSending::IDD, pParent)
{
        //{{AFX_DATA_INIT(CDlgSending)
        m_SeFile = _T("");
        //}}AFX_DATA_INIT
}

void CDlgSending::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CDlgSending)
        DDX_Text(pDX, IDC_EDIT1, m_SeFile);
        //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CDlgSending, CDialog)
        //{{AFX_MSG_MAP(CDlgSending)
        ON_BN_CLICKED(IDC_BUTTON1, OnSelect_File)
        ON_BN_CLICKED(IDOK, OnSendFile)
        //}}AFX_MSG_MAP
```

```
END_MESSAGE_MAP()
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// CDlgSending message handlers
void CDlgSending::OnSelect_File()
{
        // TODO: Add your control notification handler code here
        CString ResveFile;
        CFileDialog m_IdFile (TRUE,NULL, NULL, OFN_HIDEREADONLY,"text
        files|*.txt|encrypt file|*.cry1");
        m_IdFile.m_ofn.lpstrInitialDir = "c:\\";
        if (m_IdFile.DoModal() != IDCANCEL ){
        ResveFile = m_IdFile.GetPathName();
        m_SeFile=ResveFile;
        UpdateData(FALSE);
        }else MessageBox("Please Select File To Complete Sending
        Operation","Sending",MB_OK);
}

void CDlgSending::OnSendFile()
{
        // TODO: Add your control notification handler code here
        UpdateData(TRUE);
        if (m_SeFile !=""){
        #define port 34000
        CProgressCtrl *p=(CProgressCtrl*) GetDlgItem(IDC_PROGRESS1);
        p->SetRange(0,1000);
        p->SetPos(0);
        AfxSocketInit(NULL);
        CSocket socketSrvr;
        socketSrvr.Create (port);
        socketSrvr.Listen();
        CSocket socketRecv;
        socketSrvr.Accept(socketRecv);
```

```
        CFile myfile;
        myfile.Open (m_SeFile,CFile::modeRead|CFile::typeBinary);
        int myfileLength = myfile.GetLength ();
        socketRecv.Send(&myfileLength,4);
        byte* data = new byte[myfileLength];
        myfile.Read (data, myfileLength);
        socketRecv.Send (data, myfileLength);
        myfile.Close();
        delete data;
        socketRecv.Close();
        for(int j=1;j<1000;j++)
                p->SetPos(j);
        MessageBox("Sending Operation Sucessefully","Sending operation",MB_OK);
        p->SetPos(0);
        }else MessageBox("Sending Operation Faliled Write file name and try
again","Sending operation",MB_OK);
}

BOOL CDlgSending::OnInitDialog()
{
        CDialog::OnInitDialog();
        // TODO: Add extra initialization here
        CComQIPtr<ISkin> pSkin = GetDlgItem(IDC_SKIN21)-
        >GetControlUnknown();
        pSkin->ApplySkin((long)m_hWnd);
        return TRUE;  // return TRUE unless you set the focus to a control
        // EXCEPTION: OCX Property Pages should return FALSE
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#if
!defined(AFX_DLGSENDING_H__CB88E9AA_EE18_4264_B416_EF4ED41812F4_
_INCLUDED_)
```

```
#define
AFX_DLGSENDING_H__CB88E9AA_EE18_4264_B416_EF4ED41812F4__INCLU
DED_
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// DlgSending.h : header file
/////////////////////////////////////////////////////////////////////////////
// CDlgSending dialog
class CDlgSending : public CDialog
{
// Construction
public:
        CDlgSending(CWnd* pParent = NULL);   // standard constructor
// Dialog Data
        //{{AFX_DATA(CDlgSending)
        enum { IDD = IDD_Sending };
        CString         m_SeFile;
        //}}AFX_DATA
// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CDlgSending)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL
// Implementation
protected:
        // Generated message map functions
        //{{AFX_MSG(CDlgSending)
        afx_msg void OnSelect_File();
        afx_msg void OnSendFile();
        virtual BOOL OnInitDialog();
        //}}AFX_MSG
```

```
	DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the
previous line.
#endif //
!defined(AFX_DLGSENDING_H__CB88E9AA_EE18_4264_B416_EF4ED41812F4_
_INCLUDED_)
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// DlgRecever.cpp : implementation file
#include "stdafx.h"
#include "RC6.h"
#include "DlgRecever.h"
#include "RC6Dlg.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// CDlgRecever dialog
CDlgRecever::CDlgRecever(CWnd* pParent /*=NULL*/)
	: CDialog(CDlgRecever::IDD, pParent)
{
	//{{AFX_DATA_INIT(CDlgRecever)
	m_Ip = _T("");
	m_RFile = _T("");
	//}}AFX_DATA_INIT
}

void CDlgRecever::DoDataExchange(CDataExchange* pDX)
{
```

```
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CDlgRecever)
    DDX_Text(pDX, IDC_EDIT1, m_Ip);
    DDX_Text(pDX, IDC_EDIT2, m_RFile);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CDlgRecever, CDialog)
    //{{AFX_MSG_MAP(CDlgRecever)
    ON_BN_CLICKED(IDOK, OnReceveFile)
    ON_BN_CLICKED(IDC_BUTTON1, OnReturnBack)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// CDlgRecever message handlers
void CDlgRecever::OnReceveFile()
{
    // TODO: Add your control notification handler code here
    UpdateData(TRUE);
    if ((m_Ip !="") && (m_RFile!=""))
    {
    #define port 34000
    CProgressCtrl *p=(CProgressCtrl*) GetDlgItem(IDC_PROGRESS1);
    p->SetRange(0,1000);
    p->SetPos(0);
    AfxSocketInit(NULL);
    CSocket sockClient;
    sockClient.Create();
    sockClient.Connect(m_Ip, port);
    int dataLength;
    sockClient.Receive(&dataLength,4);
    byte* data = new byte[dataLength];
    sockClient.Receive (data, dataLength);
```

```
        CFile desFile(m_RFile,
        CFile::modeCreate|CFile::modeWrite|CFile::typeBinary);
        desFile.Write(data, dataLength);
        desFile.Close();
        delete data;
        sockClient.Close();
        for(int j=1;j<1000;j++)
        p->SetPos(j);
        MessageBox("Receving Operation Sucessefully","Receving
        operation",MB_OK);
        p->SetPos(0);
        }else MessageBox("Receiving Operation Failed Enter correct IP Address
        Destinations Location Try again ","Receiving File",MB_OK);
}

void CDlgRecever::OnCancel()
{
        // TODO: Add extra cleanup here
        CDialog::OnCancel();
}

BOOL CDlgRecever::OnInitDialog()
{
        CDialog::OnInitDialog();
        // TODO: Add extra initialization here
        CComQIPtr<ISkin> pSkin = GetDlgItem(IDC_SKIN21)-
        >GetControlUnknown();
        pSkin->ApplySkin((long)m_hWnd);
        return TRUE;  // return TRUE unless you set the focus to a control
        // EXCEPTION: OCX Property Pages should return FALSE
}
```

```
void CDlgRecever::OnReturnBack()
{
	// TODO: Add your control notification handler code here
	CDialog::OnCancel();

}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#if
!defined(AFX_DLGRECEVER_H__DED35E0D_6D79_4CA9_9F25_E63AF95CD6A
9__INCLUDED_)
#define
AFX_DLGRECEVER_H__DED35E0D_6D79_4CA9_9F25_E63AF95CD6A9__INCL
UDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// DlgRecever.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CDlgRecever dialog

class CDlgRecever : public CDialog
{
// Construction
public:
	CDlgRecever(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
	//{{AFX_DATA(CDlgRecever)
	enum { IDD = IDD_Recever };
	CString	m_Ip;
```

```
        CString         m_RFile;
        //}}AFX_DATA

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CDlgRecever)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL
// Implementation
protected:

        // Generated message map functions
        //{{AFX_MSG(CDlgRecever)
        afx_msg void OnReceveFile();
        virtual void OnCancel();
        virtual BOOL OnInitDialog();
        afx_msg void OnReturnBack();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the
previous line.
                                                                                    //
#endif
!defined(AFX_DLGRECEVER_H__DED35E0D_6D79_4CA9_9F25_E63AF95CD6A
9__INCLUDED_)
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// DecDlg.cpp : implementation file
#include "stdafx.h"
#include "RC6.h"
#include "DecDlg.h"
```

```
#include "RC6Dlg.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// CDecDlg dialog
CDecDlg::CDecDlg(CWnd* pParent /*=NULL*/)
        : CDialog(CDecDlg::IDD, pParent)
{
        //{{AFX_DATA_INIT(CDecDlg)
        m_pkey = _T("");
        m_iteration = 0;
        m_encfile = _T("");
        //}}AFX_DATA_INIT
}

void CDecDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CDecDlg)
        DDX_Text(pDX, IDC_EDIT2, m_pkey);
        DDX_Text(pDX, IDC_EDIT3, m_iteration);
        DDX_Text(pDX, IDC_EDIT1, m_encfile);
        //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CDecDlg, CDialog)
        //{{AFX_MSG_MAP(CDecDlg)
        ON_BN_CLICKED(IDC_BUTTON1, OnBrowse)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

```
// CDecDlg message handlers
void CDecDlg::OnOK()
{
	// TODO: Add extra validation here
	CDialog::OnOK();
}

void CDecDlg::OnCancel()
{
	// TODO: Add extra cleanup here
	CDialog::OnCancel();
}

void CDecDlg::OnBrowse()
{
	// TODO: Add your control notification handler code here
	CFileDialog m_IdFile (TRUE,NULL, NULL,
	OFN_HIDEREADONLY,"Encryption files|*.Cry1");
	m_IdFile.m_ofn.lpstrInitialDir = "c:\\";
	if (m_IdFile.DoModal() != IDCANCEL ){
		m_encfile = m_IdFile.GetPathName();
		decfile=(m_IdFile.GetFileTitle() + ".txt");
		UpdateData(FALSE);
	}else MessageBox("Please Select File To Complet
	Decryption","Decryption",MB_OK);
}

BOOL CDecDlg::OnInitDialog()
{
	CDialog::OnInitDialog();
	// TODO: Add extra initialization here
	CComQIPtr<ISkin> pSkin = GetDlgItem(IDC_SKIN21)-
	>GetControlUnknown();
```

```
        pSkin->ApplySkin((long)m_hWnd);
        return TRUE;  // return TRUE unless you set the focus to a control
        // EXCEPTION: OCX Property Pages should return FALSE
}
/////////////////////////////////////////////////////////////////////////////////////////////////////////////
// CDecDlg dialog
class CDecDlg : public CDialog
{
// Construction
public:
        CString decfile;
        CDecDlg(CWnd* pParent = NULL);   // standard constructor
// Dialog Data
        //{{AFX_DATA(CDecDlg)
        enum { IDD = IDD_Dec };
        CString         m_pkey;
        int             m_iteration;
        CString         m_encfile;
        //}}AFX_DATA


// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CDecDlg)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL
// Implementation
protected:

        // Generated message map functions
        //{{AFX_MSG(CDecDlg)
        virtual void OnOK();
```

```
        virtual void OnCancel();
        afx_msg void OnBrowse();
        virtual BOOL OnInitDialog();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the
previous line.
#endif                                                                          //
!defined(AFX_DECDLG_H__C134F94A_0B00_48A4_B656_F67029B015A1__INCL
UDED_)
////////////////////////////////////////////////////////////////////////////////////////////////////
BOOL CRC6Dlg::OnInitDialog()
{
        CDialog::OnInitDialog();
        // Add "About..." menu item to system menu.
        // IDM_ABOUTBOX must be in the system command range.
        ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
        ASSERT(IDM_ABOUTBOX < 0xF000);
        CMenu* pSysMenu = GetSystemMenu(FALSE);
        if (pSysMenu != NULL)
        {
                CString strAboutMenu;
                strAboutMenu.LoadString(IDS_ABOUTBOX);
                if (!strAboutMenu.IsEmpty())
                {
                        pSysMenu->AppendMenu(MF_SEPARATOR);
                        pSysMenu->AppendMenu(MF_STRING,    IDM_ABOUTBOX,
                        strAboutMenu);
                }
        }
```

```
// Set the icon for this dialog.  The framework does this automatically
//  when the application's main window is not a dialog
SetIcon(m_hIcon, TRUE);                    // Set big icon
SetIcon(m_hIcon, FALSE);           // Set small icon
// TODO: Add extra initialization here0
        m_Rad1=0;
        GetDlgItem ( IDC_BUTTON4) -> EnableWindow ( FALSE ) ;
        UpdateData(FALSE);
        CComQIPtr<ISkin> pSkin = GetDlgItem(IDC_SKIN22)-
        >GetControlUnknown();
        pSkin->ApplySkin((long)m_hWnd);
        return TRUE;  // return TRUE  unless you set the focus to a control
}

void CRC6Dlg::OnSysCommand(UINT nID, LPARAM lParam)
{
        if ((nID & 0xFFF0) == IDM_ABOUTBOX)
        {
                CAboutDlg dlgAbout;
                dlgAbout.DoModal();
        }
        else
        {
                CDialog::OnSysCommand(nID, lParam);
        }
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.
```

```
void CRC6Dlg::OnPaint()
{
        if (IsIconic())
        {
                CPaintDC dc(this); // device context for painting
                SendMessage(WM_ICONERASEBKGND, (WPARAM)
                dc.GetSafeHdc(), 0);
                // Center icon in client rectangle
                int cxIcon = GetSystemMetrics(SM_CXICON);
                int cyIcon = GetSystemMetrics(SM_CYICON);
                CRect rect;
                GetClientRect(&rect);
                int x = (rect.Width() - cxIcon + 1) / 2;
                int y = (rect.Height() - cyIcon + 1) / 2;
                // Draw the icon
                dc.DrawIcon(x, y, m_hIcon);
        }
        else
        {
                CDialog::OnPaint();
        }
}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CRC6Dlg::OnQueryDragIcon()
{
        return (HCURSOR) m_hIcon;
}
```