Comparative Cryptanalysis In Parallel Computing Environments

Developed By

Umar Waqar Anis

Khawaja Amer Hayat

Supervised By

Dr. S.Tauseef-ur-Rehman

Department of Computer Science

International Islamic University Islamabad

2004

بسم الله الرحمن الرحيم

In the name of ALMIGHTY ALLAH, The most Beneficent, the most Merciful.

Department of Computer Science,
International Islamic University, Islamabad.

Final Approval

Date 17/6/04

It is certified that we have read the thesis, titled "**Comparative Cryptanalysis in Parallel Computing Environments**" submitted by **Umar Waqar Anis** under University Reg. No. **65-CS/MS/02** and **Khuwaja Amer Hayat** under University Reg. No. **64-CS/MS/02**. It is our judgment that this thesis is of sufficient standard to warrant its acceptance by the International Islamic University, Islamabad, for the Degree of **Master of Science**.

Committee

External Examiner
Dr. Adul Satar
Project Director,
Allama Iqbal Open University, Islamabad

Internal Examiner
Mr. Asim Munir
Department of Computer Science,
International Islamic University, Islamabad.

Supervisor
Dr. Tauseef-ur-Rehman
Head,
Department of Telecommunication Engineering,
International Islamic University, Islamabad.

To our Families

A dissertation submitted to the
Department of Computer Science,
International Islamic University, Islamabad
as a partial fulfillment of the requirements
for the award of the degree of
Master of Science

Declaration

We hereby declare that this software, neither as a whole nor as a part thereof has been copied out from any source. It is further declared that we have developed this software entirely on the basis of our personal efforts made under the sincere guidance of our teachers. No portion of the work presented in this report has been submitted in support of any application for any other degree or qualification of this or any other university or institute of learning.

Umar Waqar Anis
65-CS/MS/02

Khuwaja Amer Hayat
64-CS/MS/02

Acknowledgements

First of all we are thankful to Almighty Allah (*SWT*) the creator and sustainer of this universe, without His blessings it was impossible to carry out this study successfully.

It is our proud privilege to express our gratitude and deep sense of obligation to Dr. S. Tauseef-ur-Rehman Head, Department of Telecommunication Engineering, International Islamic University, Islamabad, without whose keen and personal interest, helpful advice patience and able guidance this work would not have been in its present shape.

We feel pleasure to acknowledge the valuable support and cooperation of the faculty members.

Our special thanks are due to our brother and friend Mr. Muhammad Adeel Abduhu for his help, support and continuous encouragement.

Last but not least, we are indebted to our grand parents, parents and sisters, for their never-ending prayers, love and affection.

Umar Waqar Anis
65-CS/MS/02

Khuwaja Amer Hayat
64-CS/MS/02

Project in Brief

Project Title:	Comparative Cryptanalysis in Parallel Computing Environments
Objective:	To design and implement and compare cryptanalytic attacks on DES
Undertaken By:	**Umar Waqar Anis** **Khuwaja Amer Hayat**
Supervised By:	**Dr. Tauseef-ur-Rehman** Head, Department of Telecommunication Engineering, International Islamic University, Islamabad.
Technologies Used:	C++, Kate editor
System Used:	Pentium® IV
Operating System Used:	Redhat Linux 9.0
Date Started:	1st September, 2003
Date Completed:	15th May, 2004

Abstract

This study aims at carrying cryptanalytic attacks on DES (reduced rounds) the attacks are linear cryptanalysis, differential cryptanalysis and related key cryptanalysis. The algorithms developed for these attacks were executed in parallel computing environment (cluster) because of the time consuming nature of these attacks. The results than obtained were compared to have an approximation of the time required by each attack and hence deducing the efficient algorithm. The remainder of this document provides descriptions of the interfaces to and implementation of each of these mechanisms.

Table of Contents

List of Figures

Chapter 1

Introduction

1. Introduction

With the introduction of distributed systems and the use of networks and communication facilities for carrying data between terminal users and computers and between computer and computer, security becomes an issue. For this Network security measures are taken to protect data during its transmission.

By far the most important automated tool for network and communication security is encryption i.e. Cryptography. In the conventional methods the two parties share the same encryption/decryption key. The problem with this approach is the protection and distribution of keys. Present day systems uses a different approach in which encryption is done through a separate key (public key).

Cryptanalysis refers to the study of ciphers, cipher text, or cryptosystems with a view to finding weaknesses in them that permit retrieval of the plaintext from the cipher text, without necessarily knowing the key or the algorithm. This is known as "breaking" the cipher, cipher text, or cryptosystem.

Breaking is sometimes used interchangeably with "weakening". This refers to finding a property (fault) in the design or implementation of the cipher that reduces the number of keys required in a brute force attack .A cryptanalysis of the cipher reveals a technique that would allow the plaintext to be found. The plaintext can be found with moderate computing resources.

1.1 Cryptography

Modern cryptography is a remarkable field. It deals with very human concerns issues of privacy, authenticity, and trust. It does so in a way that is concrete and scientific.

The word cryptography comes from the Latin word "crypt" meaning "secret" and "graphia" meaning writing. So cryptography is literally "secret writing".

Modern cryptography is the science of using mathematics to protect data. Cryptography ensures that sensitive information cannot be read by anyone except the intended recipient.

The process of converting the data into some unreadable/non understandable form is termed as encryption, and the reverse process is termed as decryption.

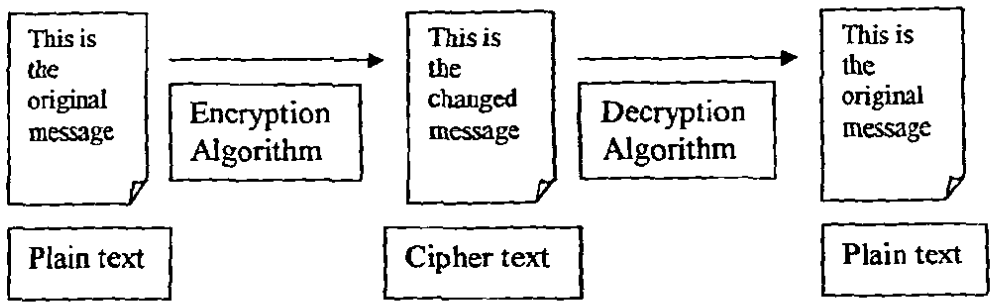


Figure 1 Cryptography

A cryptographic algorithm, or cipher, is a mathematical function used in the encryption and decryption process. A cryptographic algorithm works in combination with a key (word, number or phrase) to encrypt the plaintext. The same plaintext encrypts to different cipher text with different keys. The security of encrypted data is entirely dependent on two things:

- Strength of the cryptographic algorithm
- Secrecy of the key.

A cryptographic algorithm, plus all possible keys and all the protocols that make it work, comprise a cryptosystem. DES, Triple DEA, AES and PGP are examples of cryptosystems.

When a plain text/ original message M undergoes an encryption function E it produces out a cipher text C.

$$E(M) = C \quad \ldots\ldots(1)$$

In the reverse process, the decryption function D operates on C to produce M:

$$D(C) = M \quad \ldots\ldots(2)$$

Since the whole point of encrypting and then decrypting a message is to recover the original plaintext, the following identity must hold true:

$$D(E(M)) = M \quad \ldots\ldots(3)$$

1.2 Keys

A key K is a value that works with a cryptographic algorithm to produce a specific cipher text. Keys are big numbers. Key size is measured in bits. In public-key cryptography, the bigger the key, the more secure the cipher text. However, public key size and conventional cryptography's secret key size are totally unrelated. A conventional 80-bit key has the equivalent strength of a 1024-bit public key. A conventional 128-bit key is equivalent to a 3000-bit public key. Again, the bigger the key, the more secure, but the algorithms used for each type of cryptography are very different and thus cannot be compared.[BS93].However if the underlying pattern is identified than key size is irrelevant.

So in terms of keys our Eq (1) becomes

$$E_K(M) = C \quad \ldots\ldots(4)$$

The Decryption algorithm works in reverse of the encryption algorithm. It takes C and k as input and produces M.

$$D_K(C) = M \quad \ldots\ldots(5)$$

$$D_K(E_K(M)) = M \quad \ldots\ldots(6)$$

Some algorithms use a different encryption key and decryption key. That is, the encryption key, K_1, is different from the corresponding decryption key, K_2. In this case

$$E_{K1}(M) = C \quad \ldots\ldots(7)$$

$$D_{K2}(C) = M \quad \ldots\ldots(8)$$

$$D_{K2}(E_{K1}(M)) = M \quad \ldots\ldots(9)$$

The public and private keys (K_1 and K_2) are mathematically related and it is very difficult to derive the private key given only the public key; however, deriving the public key is always possible given enough time and computing power. This makes it very important to pick keys of the right size; large enough to be secure, but small enough to be applied fairly quickly. Larger keys will be cryptographically secure for a longer period of time.[BS93]

1.3 Classification of Cryptosystems

The different cryptosystems are usually classified into the following categories. They are as fallow:

- Conventional/Symmetric Cryptosystems

- Public Key/Asymmetric Cryptosystems

1.3.1 Conventional/Symmetric Cryptosystems

In conventional cryptography, also called secret-key or symmetric-key encryption, one key is used both for encryption and decryption. The Data Encryption Standard (DES) in an example of a conventional cryptosystem. The following figure is an illustration of the conventional encryption process.

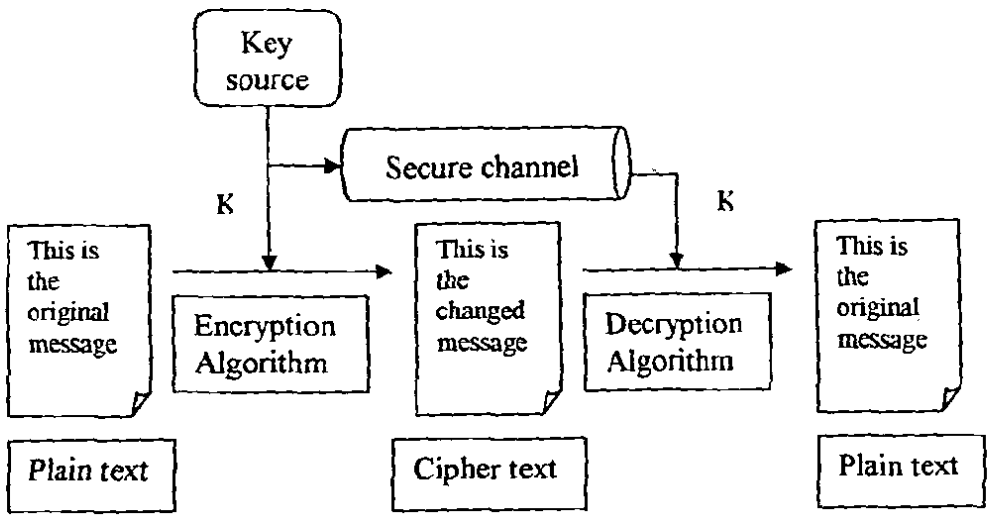


Figure 2 Symmetric Cryptosystems

Conventional encryption has some benefits. However, conventional encryption alone as a mean for transmitting secure data can be quite expensive simply due to the difficulty of secure key distribution.

1.3.2 Public key/Asymmetric Cryptosystem

Public-key cryptography uses a pair of keys, a public key, which encrypts data, and a corresponding private key, for decryption. Because it uses two keys, it is sometimes called asymmetric cryptography.

It is computationally infeasible to deduce the private key from the public key. Anyone who has a public key can encrypt information but cannot decrypt it.

The primary benefit of public-key cryptography is that it allows people who have no preexisting security arrangement to exchange messages securely. The need for sender and

receiver to share secret keys via some secure channel is eliminated all communications involve only public keys, and no private key is ever transmitted or shared. Some examples of public-key cryptosystems are Elgamal (named for its inventor, Taher Elgamal), RSA (named or its inventors, Ron Rivest, Adi Shamir, and Leonard Adleman), Diffie-Hellman , and DSA, the Digital Signature Algorithm, (invented by David Kravitz)

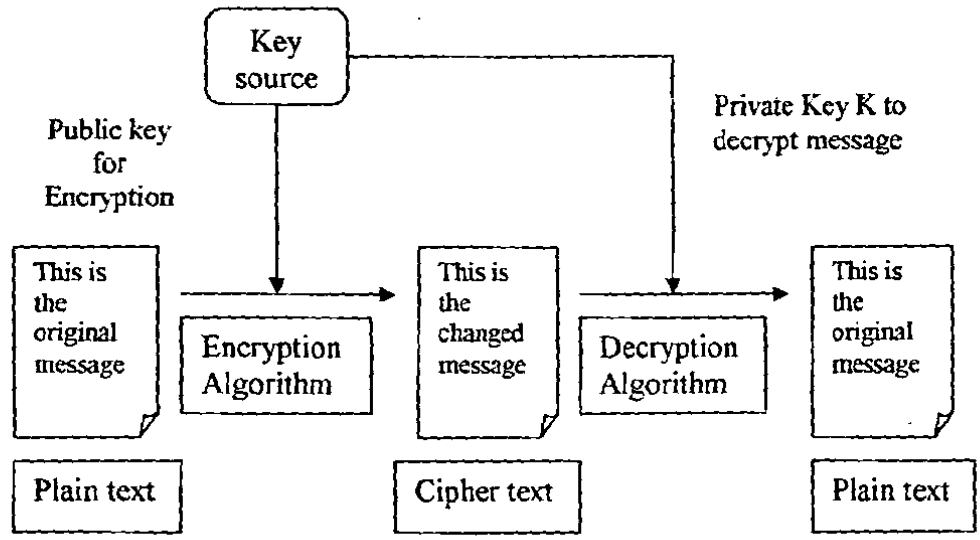


Figure 3 Public key Cryptosystems

1.4 Types of Ciphers

A cipher can be taken as an algorithm that takes plain text as input and perform some operations on it and than generates a cipher text. The cipher text is usually created by three operations on plaintext. These operations are

- Substitutions
- Permutations
- Logic Operations

Substitutions: In substitutions the characters of the original text are substituted with some fake/ other characters.

Permutations: In Permutations the characters with in the text are interchanged.

Logic Operations: Some Mathematical logic operators (such as XOR) are applied on the plaintext

The ciphers are of two types

- Block Cipher
- Stream Cipher

1.4.1 Block Ciphers

A block cipher operates on a block of plain text at a time. This block is completely processed at one time.

1.4.2 Stream Ciphers

Stream ciphers convert plaintext to cipher text 1 bit at a time. A key stream generator (sometimes called a running-key generator) outputs a stream of bits K_1, K_2, K_3,..., K_i. This key stream (sometimes called a running key) is XORed with a stream of plaintext bits, P1, P2, P3,..., Pi, to produce the stream of cipher text bits.

$$C_i = P_i \oplus Ki$$

At the decryption end, the cipher text bits are XORed with an identical key stream to recover the plaintext bits.

$$P_i = C_i \oplus K_i$$

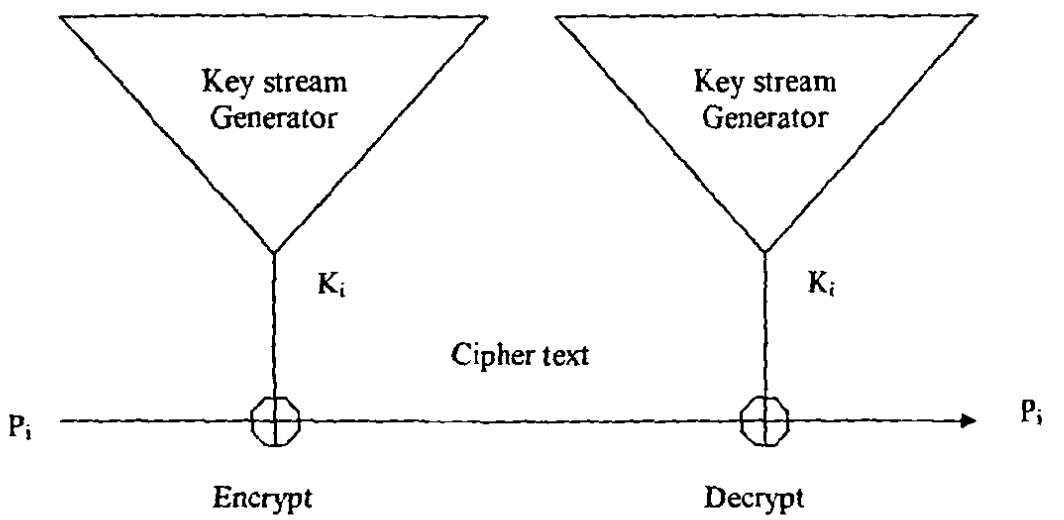


Figure 4 Stream Ciphers

The system's security depends entirely on the insides of the key stream generator. If the key stream generator outputs an endless stream of zeros, the cipher text will equal the plaintext and the whole operation will be worthless. If the key stream generator spits out a repeating 16-bit pattern, the algorithm will be a simple XOR with negligible security.

The reality of stream ciphers security lies somewhere between the simple XOR and the one-time pad. The key stream generator generates a bit stream that looks random, but is

actually a deterministic stream that can be flawlessly reproduced at decryption time.[BS93]

Most stream ciphers have keys. The output of the key stream generator is a function of the key. A key stream generator has three basic parts. The internal state describes the current state of the key stream generator. Two key stream generators, with the same key and the same internal state, will produce the same key stream. The output function takes the internal state and generates a key stream bit. The next-state function takes the internal state and generates a new internal state.

1.5 Modes of Operations

A mode can be defined as a sequence in which cryptographic functions are performed. A cryptographic mode usually combines the basic cipher, some sort of feedback, and some simple operations. The operations are simple because the security is a function of the underlying cipher and not the mode. Even more strongly, the cipher mode should not compromise the security of the underlying algorithm.

There are other security considerations. Patterns in the plaintext should be concealed, input to the cipher should be randomized, manipulation of the plaintext by introducing errors in the cipher text should be difficult, and encryption of more than one message with the same key should be possible. Efficiency is another consideration. The mode should not be significantly less efficient than the underlying cipher. In some circumstances it is important that the cipher text be the same size as the plaintext.

A third consideration is fault-tolerance. Some applications need to parallelize encryption or decryption, while others need to be able to preprocess as much as possible. In still others it is important that the decrypting process be able to recover from bit errors in the cipher text stream, or dropped or added bits. Different cryptographic modes have different subsets of these characteristics.

- Electronic Codebook Mode
- Cipher Block Chaining Mode
- Cipher-Feedback Mode
- Output-Feedback Mode

- Counter Mode

1.5.1 Electronic Code Book

Electronic codebook (ECB) mode is the most obvious way to use a block cipher. A block of plaintext encrypts into a block of cipher text. Since the same block of plaintext always encrypts to the same block of cipher text, it is theoretically possible to create a code book of plaintexts and corresponding cipher texts. However, if the block size is 64 bits, the code book will have 2^{64} entries much too large to pre compute and store. Furthermore, every key has a different code book.

1.5.2 Cipher Block Chaining Mode

Chaining adds a feedback mechanism to a block cipher. The results of the encryption of previous blocks are fed back into the encryption of the current block. Each cipher text block is dependent not just on the plaintext block that generated it but on all the previous plaintext blocks.

In cipher block chaining (CBC) mode, the plaintext is XORed with the previous cipher text block before it is encrypted.

A cipher text block is decrypted normally and also saved in a feedback register. After the next block is decrypted, it is XORed with the results of the feedback register. Then the next cipher text block is stored in the feedback register, and so on, until the end of the message.

Mathematically, this looks like

$$C_i = E_K(P_i \oplus C_{i-1})$$

$$P_i = C_{i-1} \oplus D_K(C_i)$$

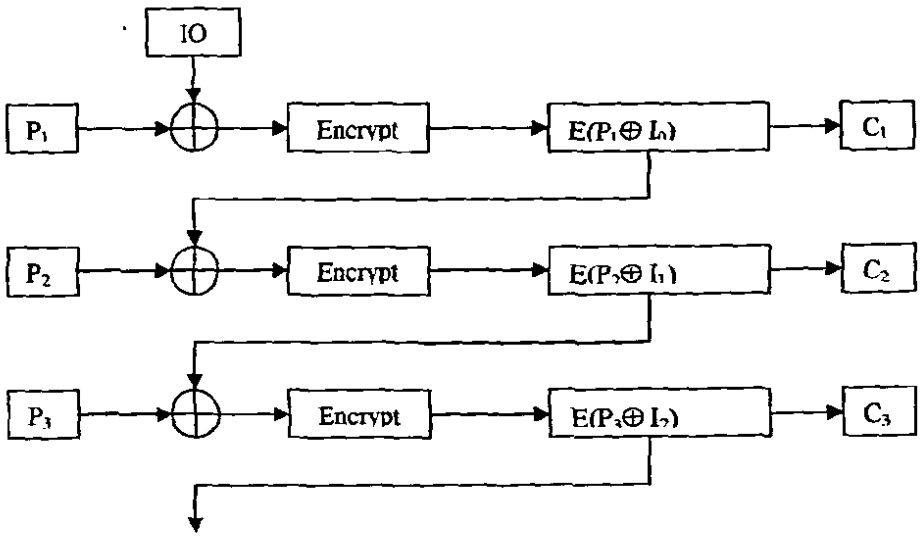


Figure 5 Cipher Block Chaining

1.5.3 Cipher Feed Back Mode

In CFB mode, data can be encrypted in units smaller than the block size. CFB links the plaintext, so that the cipher text depends on the preceding plaintext.

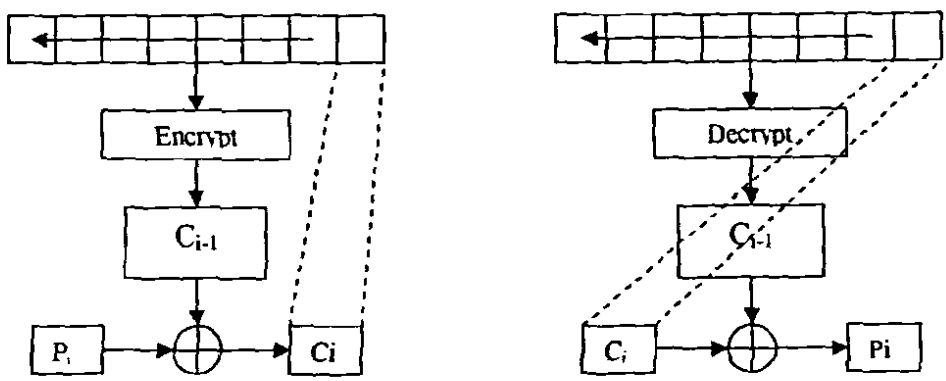


Figure 6 Cipher Feed Back Mode

1.5.4 Output Feedback Mode

Output Feedback Mode is a method of running a block cipher as a synchronous stream cipher. It is similar to CFB mode, except that n bits of the previous output block are moved into the right-most positions of the queue. Decryption is the reverse of this process. This is called n-bit OFB. On both the encryption and the decryption sides, the block algorithm is used in its encryption mode. This is sometimes called internal feedback, because the feedback mechanism is independent of both the plaintext and the cipher text streams.

$$C_i = P_i \oplus S_i; \quad S_i = E_K(S_{i-1})$$

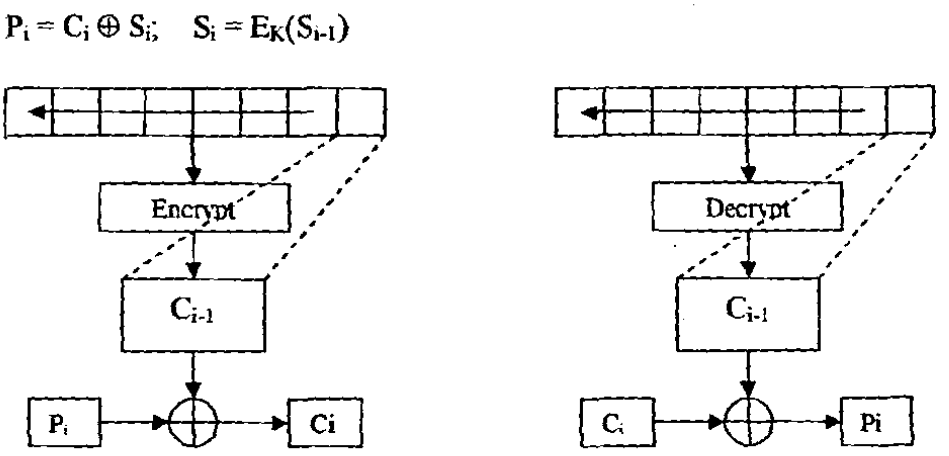


Figure 7 Output Feed Back Mode

1.5.5 Counter Mode

Block ciphers in counter mode use sequence numbers as the input to the algorithm. Instead of using the output of the encryption algorithm to fill the register, the input to the register is a counter. After each block encryption, the counter increments by some constant, typically one. The synchronization and error propagation characteristics of this mode are identical to those of OFB.

1.6 Digital Signature

Public key cryptography is that it provides a method for employing digital signatures. Digital signatures let the recipient of information verify the authenticity of the information's origin, and also verify that the information was not altered while in transit. Thus, public key digital signatures provide authentication and data integrity.

A digital signature serves the same purpose as a seal on a document, or a handwritten signature. However, because of the way it is created, it is superior to a seal or signature in an important way. A digital signature not only attests to the identity of the signer, but it also shows that the contents of the information signed have not been modified. A physical seal or handwritten signature cannot do that. However, like a physical seal that can be created by anyone with possession of the signet, a digital signature can be created by anyone with the private key of that signing key pair.

The signature algorithm uses private key to create the signature and the public key to verify it.

1.7 Hash functions

The system described above has some problems. It is slow, and it produces an enormous volume of data at least double the size of the original information. An improvement on the above scheme is the addition of a one-way hash function in the process. A one-way hash function takes variable-length input in this case, a message of any length, even thousands or millions of bits and produces a fixed-length out-put say, 160 bits. The hash function ensures that, if the information is changed in any way even by just one bit an entirely different output value is produced.

1.8 One Time Pad

One-time pad is a perfect encryption scheme [BS93], and was invented in 1917 by Major Joseph Mauborgne and AT&T's Gilbert Vernam [Khan67]. A one-time pad is a special case of a threshold scheme. Classically, a one-time pad is nothing more than a large non repeating set of truly random key letters, written on sheets of paper, and glued together in a pad. In its original form, it was a one-time tape for teletypewriters. The sender uses each key letter on the pad to encrypt exactly one plaintext character. Encryption is the addition modulo 26 of the plaintext character and the one-time pad key character.

Each key letter is used exactly once, for only one message. The sender encrypts the message and then destroys the used pages of the pad or used section of the tape. The receiver has an identical pad and uses each key on the pad, in turn, to decrypt each letter of the cipher text. The receiver destroys the same pad pages or tape section after decrypting the message. For example, if the message is:

ONETIMEPAD

and the key sequence from the pad is

TBFRGFARFM

then the cipher text is

IPKLPSFHGQ

because

O + T mod 26 = I

N + B mod 26 = P

E + F mod 26 = K

1.9 Cryptanalysis

Cryptanalysis is an approach to attack/break conventional encryption. According to the scheme called a one-time pad, there is no encryption algorithm that is unconditionally secure. The process of attempting to discover X or K(encryption X using key K) or both is called cryptanalysis[BS93]. The strategy used by the cryptanalyst depends upon the nature of the encryption scheme and the information available to the cryptanalyst. (Refer to A-1 and A–2 in Appendix-A for details). Some popular forms of cryptanalysis are:

- Linear Cryptanalysis
- Differential Cryptanalysis
- Quantum Cryptanalysis
- Related Key Cryptanalysis
- Quadratic Cryptanalysis

The whole point of cryptography is to keep the plaintext (or the key, or both) secret from eavesdroppers (also called adversaries, attackers, interceptors, interlopers, intruders, opponents, or simply the enemy). Eavesdroppers are assumed to have complete access to the communications between the sender and receiver.

Cryptanalysis is the science of recovering the plaintext of a message without access to the key. Successful cryptanalysis may recover the plaintext or the key. It also may find weaknesses in a cryptosystem that eventually lead to the previous results. (The loss of a key through non cryptanalytic means is called a compromise.). An attempted cryptanalysis is called an attack.

There are five general types of cryptanalytic attacks. Each of them assumes that the cryptanalyst has complete knowledge of the encryption algorithm used.

- Cipher text only attacks
- Known plaintext attacks
- Chosen plaintext attacks
- Adaptive chosen plaintext attack
- Chosen cipher text attack

1.9.1 Cipher text-only attack

The cryptanalyst has the cipher text of several messages, all of which have been encrypted using the same encryption algorithm. The cryptanalyst's job is to recover the plaintext of as many messages as possible, or better yet to deduce the key (or keys) used to encrypt the messages, in order to decrypt other messages encrypted with the same keys.

Given: $C_1 = E_k(P_1)$, $C_2 = E_k(P_2)$,...$C_i = E_k(P_i)$

Deduce: Either P_1, P_2,...P_i; k; or an algorithm to infer P_{i+1} from $C_{i+1} = E_k(P_{i+1})$

1.9.2 Known-plaintext attack

The cryptanalyst has access not only to the cipher text of several messages, but also to the plaintext of those messages. His job is to deduce the key (or keys) used to encrypt the messages or an algorithm to decrypt any new messages encrypted with the same key (or keys).

Given: P_1, $C_1 = E_k(P_1)$, P_2, $C_2 = E_k(P_2)$,...P_i, $C_i = E_k(P_i)$

Deduce: Either k, or an algorithm to infer P_{i+1} from $C_{i+1} = E_k(P_{i+1})$

1.9.3 Chosen-plaintext attack

The cryptanalyst not only has access to the cipher text and associated plaintext for several messages, but he also chooses the plaintext that gets encrypted. This is more powerful than a known-plaintext attack, because the cryptanalyst can choose specific plaintext blocks to encrypt, ones that might yield more information about the key. His job is to deduce the key (or keys) used to encrypt the messages or an algorithm to decrypt any new messages encrypted with the same key (or keys).

Given: P_1, $C_1 = E_k(P_1)$, P_2, $C_2 = E_k(P_2)$,...P_i, $C_i = E_k(P_i)$, where the cryptanalyst gets to choose P_1, P_2,...P_i

Deduce: Either k, or an algorithm to infer P_{i+1} from $C_{i+1} = E_k(P_{i+1})$

1.9.4 Adaptive-chosen-plaintext attack

This is a special case of a chosen-plaintext attack. Not only can the cryptanalyst choose the plaintext that is encrypted, but he can also modify his choice based on the results of previous encryption. In a chosen-plaintext attack, a cryptanalyst might just be able to choose one large block of plaintext to be encrypted; in an adaptive-chosen-plaintext

attack he can choose a smaller block of plaintext and then choose another based on the results of the first, and so forth.

1.9.5 Chosen cipher text attack

Cryptanalysts can choose different cipher texts to be decrypted and have access to decrypted plaintext. In an instance cryptanalyst have a tamperproof box that does automatic decryption, the job is to deduce the key.

Given: C_1, $P_1=D_K(C_1)$, $D_K(C_2)$.....$D_K(C_i)$

Deduce: K

This type of attack is preliminary applicable to public key cryptosystems. A chosen key attack also works against symmetric algorithm, but due to the symmetry of these cryptosystems it is equivalent in complexity to a chosen plain text attack

1.10 Cluster Computing

A cluster is a type of parallel or distributed processing system, which consists of a collection of interconnected stand-alone computers working together as a single, integrated computing resource.

A computer node can be a single or multiprocessor system with memory, I/O facilities, and an operating system. A cluster generally refers to two or more computers connected together. The nodes can exist in a single cabinet or be physically separated and connected via a LAN. An interconnected (LAN-based) cluster of computers can appear as a single system to users and applications. Such a system can provide a cost effective way to gain features and benefits (fast and reliable services) that have historically been found only on more expensive proprietary shared memory systems.

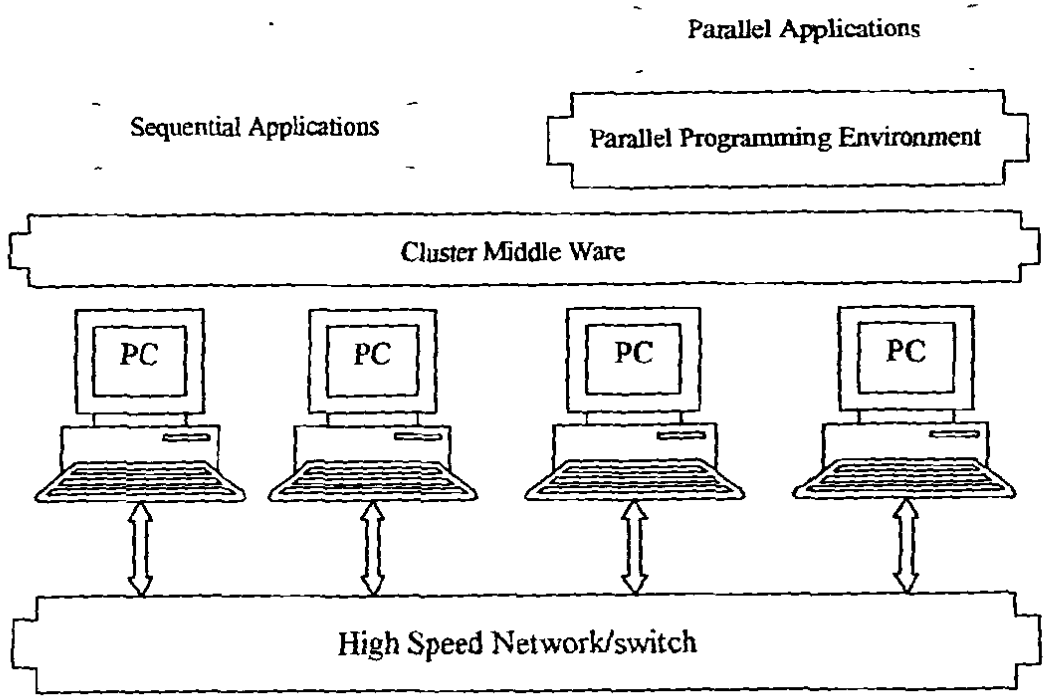


Figure 8 Cluster Architecture

The key benefits that cluster provide are:[STACA]

- Absolute Scalability
- Incremental Scalability
- High Availability
- Performance

(Refer to glossary)

The clusters can formed using different methodologies, they are

- Passive Standby
- Active Secondary
- Separate Servers
- Servers Connected to Disks
- Servers Share Disks

Chapter 2

Review of Literature

2. Review of Literature

2.1 S-Box Design

Many block ciphers are based on the old Shannon idea of the sequential application of confusion and diffusion. Confusion is provided by some form of substitution S-boxes. A substitution can be better than the other because one possible substitution maps every value onto itself.[CBT]

The hunt was on for measures which would distinguish between bad and good substitutions, and for techniques to construct good substitutions. But since weakness measures are related to attacks, new attacks often imply a need for new measures. And since it is not known that what attack is used, therefore it is difficult to select a substitution that will stand the attack.

The S boxes are expected to have average strength against unknown attacks. Where there is no systematic design, there can be no systematic weakness and when S-boxes are chosen at random, there are no S-box trap doors. Keying the S-boxes inevitably takes time, but it is considered as an advantage in slowing attacks

[Feistel73] It was 1973 when Feistel gave the concept of avalanche. In surprisingly timeless comments, he does this in the context of trying to protect individual privacy. If there are 128 inputs and outputs, for example, an analyst would have to cope with 2^{128} (or more than 10^{38}) possible digit blocks, a number so vast that frequency analysis would no longer be feasible. Unfortunately a substitution device with 128 inputs would also require 2^{128} internal terminals between the first and second switch, a technological impossibility. This is a fundamental dilemma in cryptography. As the input moves through successive layers the pattern of 1's generated is amplified and results in an unpredictable avalanche. In the end the final output will have, on average, half 0's and half 1's. The important fact is that all output digits have potentially become very involved functions of all input digits.

[KamDavid79] In 1979 Kam and Davida gave the concept of completeness. They were concerned with the particular structure which we now call a Substitution - Permutation (S-P) cipher. But while completeness is verifiably important in S-P ciphers, it may not be equally important in other ciphering structures. They defined completeness as fallow:
Definition: Give a one-one correspondence f:$\{0,1\}^n$ to $\{0,1\}^n$, f is said to be complete if, for every i,j in {1,...,n}, there exist two n-bit vectors X_1, X_2 such that X_1 and X_2 differ only in the ith bit and $f(X_1)$ differs from $f(X_2)$ in at least the jth bit.

[GordonRetkin82] Gordon and Retkin count the number of randomly chosen S-boxes which contain linear relationships. These results were updated by Youssef and Tavares [YoussefTavares95A].

[Ayoub82] Ayoub suggested S-P cipher where even the permutation is chosen at random as a way to assure users that there is no back door.

[WebsterTavares85] In 1985 Webster and Tavares reviewed completeness and avalanche and gave the Strict Avalanche Criterion (SAC).

Completeness: If a cryptographic transformation is complete, then each cipher text bit must depend on all of the output bits. Thus, if it were possible to find the simplest Boolean expression for each cipher text bit in terms of the plaintext bits, each of those expressions would have to contain all of the plaintext bits if the function was complete. Alternatively, if there is at least one pair of n-bit plaintext vectors X and X_i that differ only in bit i, and f(X) and $f(X_i)$ differ at least in bit j for all $\{ (i,j) \mid 1 <= i,j <= n \}$ then the function f must be complete

Avalanche: For a given transformation to exhibit the avalanche effect, an average of one half of the output bits should change whenever a single input bit is complemented. In order to determine whether a given m x n (m input bits and n output bits) function f satisfies this requirement, the 2^m plaintext vectors must be divided into 2^{m-1} pairs, X and X_i, such that X and X_i differ only in bit i. Then the 2^{m-1} exclusive-or sums $V_i = f(X)$ XOR $f(X_i)$ must be calculated. These exclusive-or sums will be referred to as avalanche vectors, each of which contains n bits, or avalanche variables.

Strict Avalanche Criterion: The concepts of completeness and the avalanche effect can be combined to define a new property which we shall call the strict avalanche criterion. If a cryptographic function is to satisfy the strict avalanche criterion, then each output bit should change with a probability of one half whenever a single input bit is complemented. A more precise definition of the criterion is as follows. Consider X and X_i, two n-bit binary plaintext vectors, such that X and X_i differ only in bit i, $1 <= i <= n$. Let $V_i = f(Y) \oplus f(Y_i)$

where $Y = f(X)$, $Y_i = f(X_i)$ and f is the cryptographic transformation under consideration. If f is to meet the strict avalanche criterion, the probability that each bit in V_i is equal to 1 should be one half over the set of all possible plaintext vectors X and X_i. This should be true for all values of i.

[PieprzykFinkelstein88] In 88 Pieprzyk and Finkelstein discussed the expected nonlinearity of S-boxes chosen at random.

[Forre88] Forre related strict avalanche to the Walsh spectrum, for easier testing. A necessary and sufficient condition on the Walsh-spectrum of a boolean function is given, which implies that this function fulfills the Strict Avalanche Criterion.

[MeierStaffelbach89] Meier and Staffelbach gave the idea of perfect nonlinearity and relate this to diffusion in terms of the strict avalanche criterion. With respect to linear structures, a function f has optimum nonlinearity if for every nonzero vector $\underline{a}$ in $GF(2)^n$ the values $f(\underline{x}+\underline{a})$ and $f(\underline{x})$ are equal for exactly half of the arguments $\underline{x}$ in $GF(2)^n$. If a function satisfies this property then it is perfect nonlinear with respect to linear structures, or briefly perfect nonlinear.

[Pieprzyk89A] Pieprzyk gave the error propagation property, a measure related to the SAC. In this work, indicators of the error propagation property for both Boolean functions and permutations were introduced they were examined for their natural boundaries.

[PieprzykFinkelstein89]In 89 Pieprzyk and Finkelstein dealt with the design and construction of non-linear permutations S-boxes.

[Pieprzyk89B] Pieprzyk discussed the nonlinearity of exponent permutations.

[Lloyd90]Lloyd investigates connections between the SAC, balance, and correlation immunity.

[PLLGVandewalle90] Preneel, Van Leekwijck, Van Linden, Govaerts and Vandewalle generalized the SAC and perfect nonlinearity in a Propagation Criterion of degree k. The Walsh-Hadamard transform is used.

[Nyberg91] Nyberg gave perfect nonlinearity and a construction for such S-boxes.

[DawsonTavares91] Dawson and Tavares introduced new set of S-box design criteria based on information theory.

[SivaTavPeppard92] Sivabalan, Tavares and Peppard discuss the information leakage in S-boxes, and also S-P ciphers.

[Adams92] Adams proposed to use bent functions in S-boxes.

[Cusick93] Cusick worked on counting the number of functions which satisfy the SAC of order n-4. This is related to the probability that a random selection will have the given SAC level.

[Connor93] O'Connor examined the expected Differential Cryptanalysis effects of random S-box selection.

[DaeGovVande94] Daemen, Govaerts and Vandewalle introduced the correlation matrix of a Boolean mapping which is said to be the natural representation for the proper understanding and description of the mechanisms of linear cryptanalysis.

[YouTavMisAdam95] Youssef, Tavares, Mister and Adams gave the idea of expected nonlinearity of a randomly selected injective substitution box.

[Youssef Tavares95A] Youssef and Tavares give us the probability of choosing an affine S-box. Nonlinearity is a crucial requirement for the substitution boxes in secure block ciphers. They calculated the probability of linearity in any nonzero linear combination of the output coordinates of a randomly selected regular substitution box.

[Youssef Tavares95B] Youssef and Tavares discusses the immunity of randomly selected S-boxes to differential cryptanalysis and linear cryptanalysis.

[Youssef Tavares95C] Youssef and Tavares discuss the information leakage of randomly selected functions.

[ZhangZheng95] Zhang and Zheng review the SAC and propagation criterion, and introduce their global avalanche characteristic GAC.

[Vaudenay95] Vaudenay said that S-box linearity is not so important. He applied another statistical attack the X^2-cryptanalysis without a definite idea of what happens in the encryption process. It appeared to be roughly as efficient as both differential and linear cryptanalysis.

2.2 Linear Cryptanalysis

Linear Cryptanalysis starts by finding approximate linear expressions for S-boxes and then extends these expressions to the entire cipher. Clearly, if the expressions were precisely linear, known-plaintext could immediately be solved for key bits.

Since the expressions are only approximate, in each expression a particular value for a key bit may only be slightly more probable than its complement. Accordingly, considerable known-plaintext is required before key bit values are clearly indicated.

The question for new cipher designs is whether is it ever possible to prove that no approximate linear expression exists which is sufficiently effective as to expose the key.

One answer to this is to key the S-boxes, thus depriving the analyst of precise knowledge of their contents which means that they cannot be reasonably approximated.

[Matsui93] Matsui introduced a new method for cryptanalysis of DES cipher, which was essentially a known-plaintext attack. As a result, it was possible to break 8-round DES cipher with 2^{21} known-plaintexts and 16-round DES cipher with 2^{47} known-plaintexts.

[MatsuiYamagishi94] In 1994 Matsui and Yamagishi proposed a new technique of a known plaintext attack of FEAL cipher. It was kind of meet-in-the-middle attack with a partial exhaustive key search, and therefore it derived all possible key candidates directly and deterministically. There was a checking function and a cutting off method. The former is a function g(P,C,K) whose value is constant if and only if the key candidate K satisfies Encryption(P,K) = (C) for any plaintext P and the corresponding cipher text C, and the latter is a technique to reduce the number of key candidates K

[Matsui94] Matsui gives an actual experimental cryptanalysis of DES. This was an improved version of linear cryptanalysis and its application to the first successful computer experiment in breaking the full 16-round DES. He introduced two viewpoints one was linear approximate equations based on the best (n-2)-round expression, and the other was reliability of the key candidates derived from these equations. The former reduces the number of required plaintexts, whereas the latter increases the success rate of our attack.

[DaeGovVande94] Daemen, Govaerts and Vandewalle introduce the correlation matrix of a Boolean mapping which is said to be the natural representation for the proper understanding and description of the mechanisms of linear cryptanalysis.

[KaliskiRobshaw94] Kaliski and Robshaw introduced a form of linear cryptanalysis using multiple linear approximations.

[YouTavMisAdam95] Youssef, Tavares, Mister and Adams talked about the expected nonlinearity of a randomly selected injective substitution box.

[YoussefTavares95B] Youssef and Tavares discussed the immunity of randomly selected S-boxes to differential cryptanalysis and linear cryptanalysis.

[Vaudenay95] Vaudenay says that S-box linearity is not so important.

[HarpeKraMassey95] Harpes, Kramer and Massey argued that Matsui's linear cryptanalysis for iterated block ciphers is generalized by replacing his linear expressions with I/O sums. For a single round, an I/O sum is the XOR of a balanced binary-valued function of the round input and a balanced binary-valued function of the round output. A cipher contrived to be secure against linear cryptanalysis but vulnerable to this generalization of linear cryptanalysis is given. Finally, it is argued that the ciphers IDEA and SAFER K-64 are secure against this generalization

[ButtyanVajda95] Buttyan and Vajda showed that the problem of searching for the best characteristic in linear cryptanalysis is equivalent to searching for the maximal weight path in a directed graph.

[Fauza00] In year 2000 Fauza Mirzan presented paper in which Linear behaviour was examined.

[Howard00]Howard M Haze presented a paper in which he discussed that if there is a high probability bias than the cipher is not sufficiently random.

2.3 Differential Cryptanalysis

Differential Cryptanalysis covers a growing variety of attacks on various block ciphers. It appears to be most useful on iterative (round-based) ciphers, perhaps because these can only weakly diffuse the transformations which occur in later rounds. Differential Cryptanalysis is normally a defined-plaintext attack.

The basic idea of Differential Cryptanalysis is to first cipher some plaintext, then make particular changes in that plaintext and cipher it again. Particular cipher text differences occur more frequently with some key values than others, so when those differences occur, particular keys are (weakly) indicated. With huge numbers of tests, false

indications will be distributed randomly, but true indications always point at the same key values and so will eventually rise above the noise to indicate some part of the key.

The basic concept can be applied to virtually any sort of statistic which relates cipher text changes to key values, even in relatively weak ways. But because the probabilities involved are generally quite small, success generally depends upon having very substantial amounts of known-plaintext. Thus, in practice, Differential Cryptanalysis would seem to be defeated by the simple use of message keys and limitations on the amount of material ciphered under a single message key.

Some versions [BihamShamir92] can be applied to separately-keyed blocks with a similar overall probability of success. But that success reveals only one of the many keys at random, and a success does not help with the other keys. Nor does Differential Cryptanalysis apply to message keys, since the message key value is not available as known-plaintext. Differential Cryptanalysis is powerful, but it has very significant requirements which may not be met in practice.

Differential Cryptanalysis depends upon known tables in which the key value selects various data differentials. Consequently, Differential Cryptanalysis might also be defeated by

- Keying which selects among every possible table (instead of using a few pre-defined ideal tables).
- Using data to dynamically select among a large working set of tables (instead of just four). and
- Effectively mixing table results as soon as table operations occur (rather than depending upon future round for mixing, which is risky since there are no future rounds after the last one).
- Effective mixing should prevent tables from being isolated and separately attacked.

[BihamShamir90] Biham and Shamir introduce the concept of Differential Cryptanalysis.

[Biham Shamir91] Biham and Shamir took the opportunity and breok a variety of ciphers. Two-pass Snefru was easily breakable within three minutes on a personal computer. Khafre with 16 rounds is breakable by a differential cryptanalytic chosen plaintext attack using about 1500 encryptions within about an hour on a personal computer. REDOC-II with one round is breakable by a differential cryptanalytic chosen plaintext attack using about 2300 encryptions within less than a minute on a personal computer

[BihamShamir92] Biham and Shamir attacked the Full 16-round DES. They developed an improved version of differential cryptanalysis which broke the full 16-round DES in 2^{37} time and negligible space by analyzing 2^{36} cipher texts obtained from a larger pool of 2^{47} chosen plaintexts. An interesting feature of the new attack was that it can be applied with the same complexity and success probability even if the key is frequently changed and thus the collected cipher texts are derived from many different keys.

[Nyberg Knudson92] A year after Nyberg and Knudson gave a limit for the size of the differential needed for a successful attack

[Fauza00] In year 2000 Fauza Mirzan presented a paper in which it was claimed that for an N-Round Cipher N-1 Round Differential Characteristics are to be found.

2.4 Related Key Cryptanalysis

Biham introduced the related key attack for the first time in 1994 [Biham94]. He assumes that a pair of keys has a particular relationship and the encryption is performed using these keys.

Related-key cryptanalysis assumes that the attacker learns the encryption of certain plaintexts not only under the original (unknown) key K, but also under some derived keys $K_0 = f(K)$. In a chosen-related-key attack, the attacker specifies how the key is to be changed; known-related-key attacks are those where the key difference is known, but cannot be chosen by the attacker.

Related-key cryptanalysis is a practical attack on key-exchange protocols that do not guarantee key-integrity an attacker may be able to flip bits in the key without knowing the key and key-update protocols that update keys using a known function: e.g., K, K + 1, K + 2, etc. Related-key attacks were also used against rotor machines: operators sometimes set rotors incorrectly. If the operator then corrected the rotor positions and retransmitted the same plaintext, an adversary would have a single plaintext encrypted in two related keys [DiffleHellman79].

[JohnBruceDavid98] In 1998 J Kelsey, B Schneier, D Wanger presented new related-key attacks on the block ciphers 3-WAY, Biham-DES, CAST, DES-X, New DES, RC2, and TEA. These differential related-key attacks allow both keys and plaintexts to be chosen with specific differences [JohnBruceDavid96]. These attacks show how to adapt the general attack to deal with the difficulties of the individual algorithms. In this paper specific design principles to protect against these attacks are also given.

[CietPiretJean03] In 2003 Mathieu Ciet, Gilles Piret and Jean Jacques Quisquater presented some results obtained from key schedule cryptanalysis. They dealt with related key attacks, differential related key attacks and slide attack they used slid pairs to find out a key. Furthermore, they presented sorting criteria for the selection of slid pairs.

[BellareKohno03] In year 2003 Mihir Bellare and Tadayoshi Kohno presented a paper in which they studied theoretical treatment of related key attacks. They introduced the concepts of PRPs and PRFs against classes of related key attacks. Each class was associated by a set of related key driving functions.

Chapter 3

System Analysis and Design

3. Cryptanalysis

Cryptanalysis is an approach to attack/break conventional encryption. According to the scheme called a one-time pad, there is no encryption algorithm that is unconditionally secure. The process of attempting to discover X or K (encryption X using key K) or both is called cryptanalysis. Some popular forms of cryptanalysis are:

- Differential Cryptanalysis
- Linear Cryptanalysis
- Quantum Cryptanalysis
- Related Key Cryptanalysis
- Quadratic Cryptanalysis

Below is a detailed overview of Linear, Differential and Related Key Cryptanalysis.

3.1 Differential Cryptanalysis

Differential cryptanalysis is a chosen plaintext/chosen cipher text attack that was initially developed to attack DES-like ciphers [BihamShamir90]. A chosen plaintext attack is one where the attacker is able to select inputs to a cipher and examine the output. Being one of the earlier attacks on DES, differential cryptanalysis had been studied extensively [Fauza00]. Many of today's ciphers are designed with consideration to immunity against differential cryptanalysis. Nevertheless, differential cryptanalysis still provides a good understanding of the possible weakness of ciphers and techniques to overcome them.

Differential cryptanalysis involves the analysis of the effect of the plaintext pair difference on the resulting cipher text difference. The most common difference utilized is the fixed XORed value of the plaintext pairs. By exploiting these differences, the partial subkey used in the cipher algorithm can be guessed. This guess is done statistically by using a counting procedure for each key in which the key with the highest count is assumed to be the most probable partial subkey.

3.1.1 Basic Concept

Consider the following basic linear cipher function:

$$C = P \oplus K$$

By taking the difference of a pair of cipher text, we would have cancelled out the key involved, leaving us with no information about the key:

$$C \oplus C' = P \oplus K \oplus P' \oplus K$$

$$C \oplus C' = P \oplus P'$$

This is because of the linearity of the function. The above equation simply tells us that the difference between the plaintext is the same as the difference between the cipher texts [Fauza00].

DES is not a linear cipher. Thus, the difference between the cipher texts is not equal to the difference between the plaintexts. In S-DES, the difference in a cipher text pair for a specific difference of a plaintext pair is influenced by the key. Thus, by utilizing this fact, and the knowledge that certain plaintext differences occurs with a higher probability than other differences, we can reveal information about the key [BihamShamir90]. Like linear cryptanalysis, we start by analyzing the non-linear component of the cipher, the S-Box. Then, we extend the values obtained to form a complete differential characteristic sufficient to perform an attack.

3.1.2 Difference Pair of S-Box

Consider the S-Box S_0 and S_1 of S-DES. We denote the input to the S-Box as X and the output as Y [Howard00]. The difference pairs of an S-Box is then denoted as (ΔX, ΔY), where $\Delta X = X' \oplus X''$. It is more convenient if we consider all 16 values of X' with ΔX as a constraint to the value of X'', thus $X'' = X' \oplus \Delta X$. With X' and X'', the value of ΔY can then be obtained.

The table shown in Appendix-C Table [1] shows all the difference pairs of S_0 and Appendix-C Table [2] shows the difference pairs of S_1.

3.1.3 Difference Distribution Table

The tables shown in Appendix-C Table [3] and Appendix-C Table [4] are the difference distribution tables for S_0 and S_1 where the row represents ΔX value, the column represents

ΔY values and the elements represents the number of occurrences given the column and row values. Interesting things can be performed with the availability of the difference distribution table. Two of the possibilities are:

1. We can obtain the possible input and output values given their differences [BihamShamir90].

   This is done by checking the corresponding value of the input and output differences in the difference distribution table. Consider the following input and output difference ΔX=8 and ΔY=1 of S_0. From the difference distribution table, we see that the number of occurrences for this input and output difference is 2 so only two pairs can satisfy this difference. Further, we see that these pairs are duals. If the first pair is X′, X″, then the other pair is X″, X′.

   Since ΔY is 1, then the output pairs must be 1 and 3. Subsequently, we find that the only input pairs that can yield 1 and 3 as output pairs and at the same time satisfy the input difference ΔX= 8 is 9 and 1 respectively.

2. We can obtain the key bits involved in the S-Box using known input pairs and output differences of the S-Box [BihamShamir90].

   Given a particular input pair, we can obtain the possible key bits involved in the S-Box. Assuming X′= 2, X″= 8 and the S-Box considered is S_0. Then, Y′=0, Y″=2 and ΔY = 2. We denote the inputs to the S-Box after XOR-ing with the key as I′=X′⊕ K and I″=X″⊕ K. From [Howard00], we know that the key has no influence on the input difference value. So, ΔX = ΔI = 2 ⊕ 8 = 10.

   Now that we've obtained ΔX= 10 and ΔY = 2 we can proceed to obtain the key bits involved. From the distribution table, we see that ΔX=10 and ΔY=2 has two possibilities. This implies that there are 2 possibilities for the key. The table below lists the keys and the corresponding X′ and X″. Since ΔI=10, then the pairs of I that can satisfy this difference is 7 and 13.

Given that $K = X \oplus I$, the first possible key, 5 is obtained from:

$$K = I' \oplus X' = 7 \oplus 2 = 5 \text{ and } K = I' \oplus X'' = 7 \oplus 8 = 5$$

and the second key 15 is obtained from

$$K = I'' \oplus X' = 13 \oplus 2 = 15 \text{ and } K = I'' \oplus X'' = 13 \oplus 8 = 15$$

S-Box Input	Possible Keys
7 13	5 15

Possible Keys for X'=2 and X''=8

3.1.4 Differential Characteristics

The above example is only an introduction to the possibilities that are available to us when we analyze the difference between plaintext pairs and cipher text pairs. We extend this knowledge to create a differential characteristic for 1 round of DES [Howard00]. With this differential characteristic, we can obtain the subkey, K2 used in the last round. First, we construct a differential characteristic that involves S_1 in both rounds of DES using the following difference pair of S_0 and S_1:

S_0: $\Delta X_0 = 2 \rightarrow \Delta Y_0 = 2$ with probability 12/16

S_1: $\Delta X_1 = 4 \rightarrow \Delta Y_1 = 2$ with probability 10/16

Thus, by considering ΔX_0, ΔX_1 and the expansion, E which me modify to E = [0 2 1 3 0 1 2 3], the input difference to the first round is given by:

$$\Delta U_1 = [0\,0\,0\,0\,0\,1\,0\,0]$$

The expansion E is only a form of "diffusion sugaring" and does not add to the non-linearity of the cipher. The change is to make the derivation of the input difference clearer.

Then, considering ΔY_0 and ΔY_1 and the permutation P that follows, we get the output difference for round 1:

$\Delta V_1 = [0\ 1\ 0\ 0\ 0\ 1\ 0\ 1]$

This 1-round characteristic holds with probability 12/16 x 10/16 = 15 / 32, which means that for every 32 random and uniformly distributed pairs of chosen plaintexts with difference ΔU_1 we expect to find about 1 pair of corresponding cipher texts which satisfies the difference ΔV_1. Pairs with plaintext that produces ΔU_1 and corresponding cipher texts that produces ΔV_1 are called right pairs.

The differential characteristic can be best visualized using the figure used by Biham [BihamShamir90].

For a N-round cipher, we need to find the differential characteristic of N-1 round [Howard00] [Fauza00] to conceive an attack.

3.1.5 Extracting the Partial Subkeys

With the differential characteristics obtained above, we can now proceed to extract subkey K_2 of round 2. We call the subkey that we want to extract as the target subkey [Howard00]. The process to extract the subkey is described algorithmically as follows:

1. Obtain a random plaintext P' and compute $P'' = P' \oplus \Delta X$.
2. For P' and P''.
    a. Encrypt both P' and P'' with both K_1 and K_2 to obtain C' and C''. Also, obtain CI' and CI'', the encrypted plaintext after one round, which is encrypted with K_1 only.
    b. If $CI' \oplus CI'' = \Delta Y$, then
        i. For all possible subkey values, encrypt $CI' \oplus CI''$ with only one round, which is round 2.
        ii. If the result of the encryption of 2.b.i. is equivalent to those suggested by C' and C'', then we increase the count for the corresponding subkey value used.

Repeat 1 and 2 for another random plaintext P'. Perform this until one value of the subkey has been counted to be substantially more than the others. This subkey value is assumed to be the correct subkey value used in the second round.

3.2 Linear Cryptanalysis

Linear cryptanalysis is a known-plaintext attack that is one of the most commonly used attack against block ciphers. It was invented by Mitsuru Matsui and is used initially to attack the Data Encryption Standard [Matsui94]. It is based on the fact that there are high probabilities of occurrences of linear expressions consisting the plaintext bits, cipher text bits and key bits.

3.2.1 Linear Cryptanalysis Principals

The main idea behind linear cryptanalysis is to obtain an approximation to the block cipher as a whole using a linear expression. This linear expression has the following form:

$$\left(\bigoplus_{\alpha=1}^{u} X_{i\alpha}\right) \oplus \left(\bigoplus_{\beta=1}^{v} X_{j\beta}\right) = \left(\bigoplus_{\gamma=1}^{w} X_{k\gamma}\right) \quad \text{.....(1)}$$

Where X denotes plaintext bits, Y denotes cipher text bits and K denotes the key bits. The indices u, v and w denote fixed bit locations. The goal is to find the linear expression which holds with the highest/biggest linear probability bias. The linear probability bias, e is defined as:

$$\varepsilon = \left| p - \tfrac{1}{2} \right| \quad \text{.....(2)}$$

This is the magnitude of the bias from a probability of ½. The higher the magnitude of the bias, higher will be the efficiency of the linear expression (1).

If equation (1) holds with a high probability bias, it means that the cipher used is not sufficiently random [Howard00]. A cipher is considered to be random if the randomly selected value of the bits of its linear expression would cause the expression to hold with a probability of ½. Thus, the further away a linear expression is from holding with a

probability of 0.5, the less random it is. Linear cryptanalysis takes advantage of this poor randomization.

1 1 1 1

Once a linear approximation with the highest bias is obtained, the attack can be mounted by recovering a subset of the key bits. This is done using Algorithm 1 as described in [Matsui94].

3.2.2 Obtaining a Linear Equation with High Probability Bias

To obtain a linear equation with high probability bias, we begin by constructing a statistical linear path between the input and output bits of each S-box. Then extend this path to the entire cipher and finally reached a linear expression without any intermediate value [Matsui94].

3.2.2.1 Linear Approximation of S-Boxes

Block ciphers commonly use non-linear operations in its S-boxes. Though, it is possible to construct a linear approximation of S-Boxes. Techniques for this purpose are described in [Rainier86]. The goal is to find the linear approximation with the highest bias magnitude. Following Matsui's notation,

$$NS_a(\alpha,\beta)def\#\left\{x \mid 0 \le x \le p, \left(\bigoplus_{s=0}^{y-1}(x[s]\bullet\alpha[s])\right)=\left(\bigoplus_{s=0}^{z-1}(S_a(x)[t]\bullet\beta[t])\right)\right\} \quad \ldots(3)$$

Where y is the number of input bits and z is the number of output bits.

The results of this process can be enumerated in a linear approximation table, where the vertical and the horizontal axes α and β respectively. Each element of the table represents $NS_a(\alpha,\beta) - (y + z)$. From the table, the linear approximation with the highest bias magnitude can be identified.

We now derive the linear approximation for S_0. The table in Appendix-C Table 5 shows the input to the S_0 and the corresponding output. We call this table the S_0 I/O table. The table helps to obtain the probability for a particular value of α and β. For example:

$$NS_1(1,1)=\#\left\{x \mid 0 \le x \le 166, \left(\bigoplus_{s=0}^{3}(x[s] \bullet 1[s])\right)=\left(\bigoplus_{t=0}^{1}(S_0(x)[t] \bullet 1[t])\right)\right\} \quad \text{.....(4)}$$

=8

Where Xi denotes the column to be considered as listed in the table in Appendix-C Table [5]. The above equation means that we compare the row X_0 and the row Y_0 to obtain the number of times where the element of a particular row of X_0 equals that of the same row of Y_0.

Table below shows a portion of the distribution table of S-box S_0, where the row represents α and the column represents β and the elements shows NS0(α,β) - 8. The table in Appendix-C Table [6] shows the full distribution table. Column =β

α	β		
	1	2	3
1	0	0	0
2	0	0	-2
3	0	-2	-2
4	-2	2	0
5	6	2	0
6	-2	0	-2

The most effective linear approximation is the one with the highest magnitude. We choose $NS_0(5,1)$ since $|NS_0(5,1)-8|$ is one of the highest in the table.

$$NS_0(5,1)=\#\left\{x \mid 0 \le x \le 166, \left(\bigoplus_{s=0}^{3}(x[s] \bullet 5[s])\right)=\left(\bigoplus_{t=0}^{1}(S_0(x)[t] \bullet 1[t])\right)\right\} \quad \text{.....(5)}$$

=14

Thus, the linear approximation for S_0 is $X_2 \oplus X_0 = Y_0$ which holds with probability 14/16.

3.2.2.2 Linear Approximation of the F-Function

The linear approximation of the f-function can be obtained by taking into account the expansion E and the permutation P. We extend our equation from the previous section to obtain:

$$R_0 \oplus R_2 \oplus f(R_1, K_1) = K_1 \oplus K_3 \quad \text{.....(6)}$$

3.2.2.3 Linear Approximation of the Entire Algorithm

The entire algorithm consists of two rounds. We first apply equation (6) to the first round to get the following equation:

$$R_0^1 \oplus L_0^0 \oplus R_0^0 \oplus R_2^0 = K_1^1 \oplus K_3^1 \quad \text{.....(7)}$$

The equation for the second round is:

$$L_0^1 \oplus L_0^2 \oplus R_0^1 \oplus R_2^1 = K_1^2 \oplus K_3^2 \quad \text{.....(8)}$$

Having obtained equation (7) and (8), we can now derive a linear approximation of the entire algorithm by canceling out common terms, which is:

$$L_0^0 \oplus L_0^1 \oplus L_0^2 \oplus R_0^0 \oplus R_2^0 \oplus R_2^1 = K_1^1 \oplus K_3^1 \oplus K_1^2 \oplus K_3^2 \quad \text{.....(9)}$$

We use Piling-up lemma [Matsui94] to obtain the probability that this equation hold:

$$P_T = \frac{1}{2} + 2^1 \varepsilon_1 \varepsilon_2$$

$$= 0.78125$$

3.2.3 Extracting the Partial Subkey Bits

Once a linear expression of the entire cipher had been obtained, we can deduce $K_1[1\ 3] \oplus K_2[1\ 3]$ using Algorithm 1 [Matsui94], which is as follows:

Step 1 Let T be the number of plaintexts such that the left side of equation (9) is equal to zero.

Step 2 If $T > N/2$ (N denotes the number of plaintexts),

Then

$K_1[1\ 3] \oplus K_2[1\ 3] = 0$ (when $p > 1/2$) or 1 (when $p < 1/2$)

else

$K_1[1\ 3] \oplus K_2[1\ 3] = 1$ (when $p > 1/2$) or 0 (when $p < 1/2$)

3.3 Related Key Cryptanalysis

Biham introduced the related key attack for the first time in 1994 [Biham94]. He assumes that a pair of keys has a particular relationship and the encryption is performed using these keys.

Related-key cryptanalysis assumes that the attacker learns the encryption of certain plaintexts not only under the original (unknown) key K, but also under some derived keys $K_0 = f(K)$. In a chosen-related-key attack, the attacker specifies how the key is to be changed; known-related-key attacks are those where the key difference is known, but cannot be chosen by the attacker.

Related-key cryptanalysis is a practical attack on key-exchange protocols that do not guarantee key-integrity an attacker may be able to flip bits in the key without knowing the key and key-update protocols that update keys using a known function: e.g., K, K + 1, K + 2, etc. Related-key attacks were also used against rotor machines: operators sometimes set rotors incorrectly. If the operator then corrected the rotor positions and retransmitted the same plaintext, an adversary would have a single plaintext encrypted in two related keys [DiffleHellman79].

Let K_a and K_b the two keys where

$K_a \rightarrow (K_1, K_2, \ldots\ldots, K_i)$ where $K_1, K_2, \ldots\ldots, K_i$ are subsequence of K_a and

$K_b \rightarrow (K_2, K_3, \ldots\ldots, K_i, K_1)$

The sequence of keys of K_a derives the sequence of round keys i.e. keys in K_b in other words if K_a gives rise to a sequence of sub keys, than K_b must also give rise to the same sequence of keys rotated by one round. More formally

$$K_a \rightarrow (K_1, K_2, \ldots\ldots, K_i) \leftrightarrow K_b \rightarrow (K_2, K_3, \ldots\ldots, K_i, K_1)$$

Usually the sub keys of K_b cannot be derived from K_a but this satisfies in case of LOKI Cipher attacked by Biham [Bihamn94].

Let P_a and P_b are two plain texts such that

$$P_b = F(P_a, K_1)$$

Where F(x, K) is a round function that applies to data x with round key K.
Than the encryption of plain text P_a using key K_a and that of P_b using K_b is carried out in the same way for n-1 rounds. The same property holds true for cipher texts i.e.

$$C_b = F(C_a, K_1)$$

Such a pair is known as a slid pair.
We assume that we know $2^{n/2}$ pairs of (P_a, C_a) using K_a and $2^{n/2}$ pairs of (P_b, C_b) using K_b we solve

$$F(P_a, K') = P_b$$
$$F(C_a, K') = C_b$$

For each

$$((P_a, C_a), (P_b, C_b))$$

This equation cannot hold if P_a and P_b do not form a slid pair. However when this condition is true this means that we are dealing with a slid pair and K' is actually K_1. The probability for a random pair to be a slid pair is $1/2^a$ and since there are $2.2^{n/2}$ pairs therefore we can find one. The complexity of this attack is $O(2^n)$, as $O(2^n)$ pairs are to be examined.

Chapter 4
Results

4. Results and Discussions

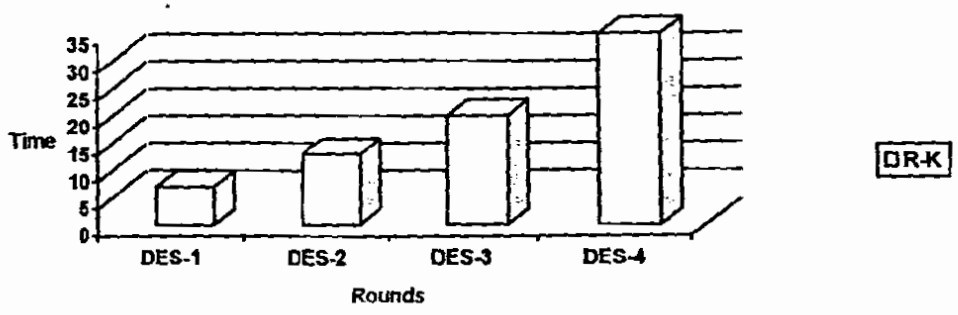
This section provides the results obtained from this study. In this study a variant of DES (reduced rounds) was subjected to different cryptanalytic attacks. The attacks were Linear cryptanalysis, differential cryptanalysis and related key cryptanalysis. These approaches have the following differences.

Linear Cryptanalysis	Differential Cryptanalysis	Related key Cryptanalysis
Linear attack focuses on trying to find out the linear expressions that hold with high probability for plaintext bits, cipher text bits and key bits.	Differential attack exploits the difference in plain text pairs and their resultant cipher text pairs.	Related key attacks use round keys on slid pairs to find out an approximate key for the given cipher.

It was observed that the related key attack produced faster results than the other two in 1st 2nd and 3rd rounds of DES.

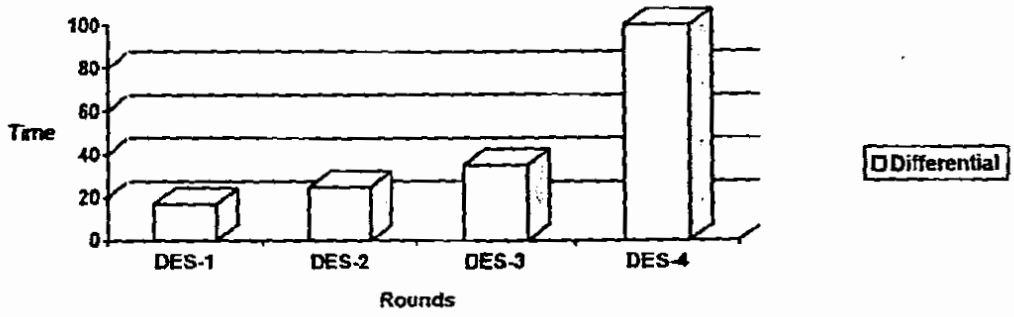
The differential attack took a little longer than Related key and Linear attacks during the first round because of the fact that it requires to generate different pairs during the first round.

The differential attack was in general was faster than Linear attack. Following is a graphical representation of the attacks.

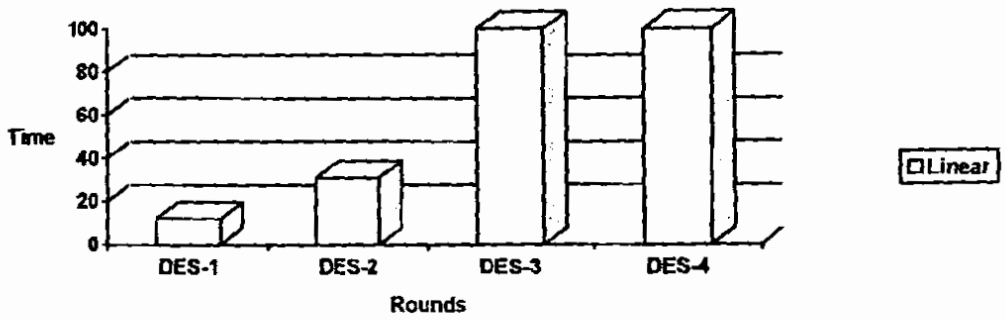


Graph showing the Time approximation for Related Key Attack.

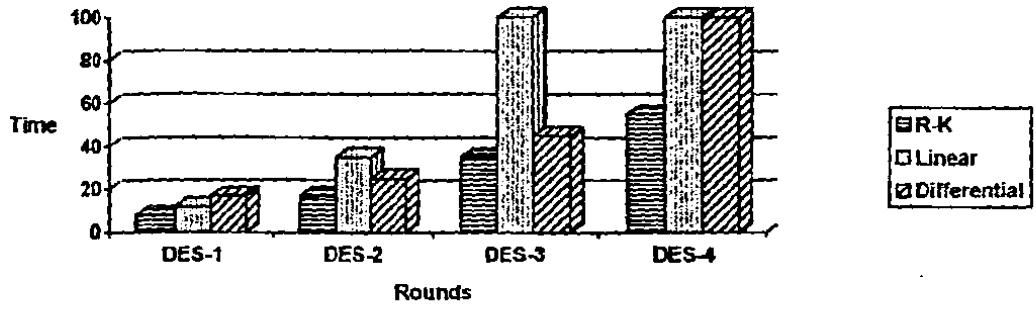
The timeline for differential attack is shown in the following graph.



The linear attack was slightly faster than the differential in the first round however, it showed the tendency to take more time in later rounds.



A comparison of all three approaches is as fallow


100
80
60
40
20
0
Time
DES-1
DES-2
DES-3
DES-4
Rounds
R-K
Linear
Differential

5. Development and Implementation

The implementation activity is the actual writing of code. If the design phase is carried out correctly, then the coding becomes a simple task. Design steps involve making the final decisions and translating the diagrams and specifications into the syntax of the chosen programming language. It also involves the practical development process, to interactively compile, link, and debug component. The work is done according to the programming rules that attempt to standardize code, developed by the programmers, and to prevent dangerous or unsuitable constructions in the language.

After the completion of the system design the following Coding activities were performed:

- Cryptology of DES algorithm
- Cryptology of TEA algorithm
- Cryptology of Blow Fish algorithm
- Linear Cryptanalysis of DES algorithm with reduced rounds
- Differential Cryptanalysis of DES algorithm with reduced rounds
- Key Related Cryptanalysis of DES algorithm with reduced rounds
- Cryptanalysis of TEA algorithm
- Cryptanalysis of Blow Fish algorithm

5.1 Environment and Tools

The development of the system was carried out using the following tools and/or technologies:

5.1.1 Redhat Linux

All products in the Red Hat Enterprise Linux family are based on a common software core--kernel, libraries, development tool chain, and utilities. This provides a homogeneous environment ideal for simplifying multi-system and desktop-to-datacenter configurations. The immediate benefits are simplified deployment of distributed applications, and a consistent environment for users and system administrators across the entire family.

- **Support for seven architectures**

  Intel X86, Intel Itanium, AMD AMD64 and IBM zSeries, iSeries, pSeries, and S/390. (With Update 2, delivered in May 2004, support for Intel EM64T was added).

- **4-4 memory split**

  Increased kernel & user address space for X86 systems, allowing support for 64GB of main memory and larger user applications.

- **Native Posix Thread Library**

  A new high-performance multi-threading capability provides improved performance for multi-threaded applications.

- **Based on Linux 2.4.21 kernel**

  Red Hat Enterprise Linux uses the latest stable Linux kernel with numerous additions from the Linux 2.5/2.6 kernels.

- **Improved scalability**

  Support for larger SMP, memory and I/O systems allows version 3 to support servers approximately twice the size of version 2.1.

- **Enhanced security**

  Includes several new security features, including support for file system ACLs.

- **Improved compiler/tools**

  Includes GCC 3.2 and debugging/profiling tools.

- **Logical Volume Manager**

  Provides enterprise-strength storage management.

- **Enhanced networking**

  Includes numerous features to improve stability & performance

5.1.2 MPICH

MPI is a library specification for message-passing. MPI was designed for high performance on both massively parallel machines and on workstation clusters. MPI is widely available, with both free available and vendor-supplied implementations. A number of MPI home pages are available.

The first standard of MPICH was presented in 1994, the Message Passing Interface (MPI) has become one of the most common API specifications for parallel programming.

MPICH is Open-Source. MPICH is the most commonly used, implementation of the MPI-1 standard

The recently arising class of parallel platforms, commonly referred to as Clusters, can only be utilized with TCP/IP as interconnect between the nodes. To use more sophisticated cluster interconnects like the Scalable Coherent Interface (SCI).

5.1.3 KDevelop

KDevelop is an easy to use IDE for developing C/C++ applications under X11.The KDevelop source is divided into several parts which correspond to subdirectories in the KDevelop project directory. There are several main parts to distinguish, mainly:

- src = The core part of KDevelop
- lib/interfaces = Plugin handler interface classes
- parts = The various parts using the KParts framework

5.1.4 QT Designer

QT Designer is a Graphical User Interface (GUI) designer toolkit which is used to add interactivity to the programs. The QT toolkit offers:

- A complete set of classes and methods ready to use even for non-graphical programming issues.
- A good solution towards user interaction by virtual methods and the signal/slot mechanism.
- A set of predefined GUI-elements, called "widgets", that can be used easily for creating the visible elements.
- Additional completely pre-defined dialogs that are often used in applications such as progress and file dialogs.

5.2 Sample Source Code

```
#define COLUMN unsigned int
#define ROW unsigned int
#define ELEMENT unsigned int
#define INDEX unsigned int
#define BYTE unsigned char
#define UINT unsigned int
#define BYTESIZE CHAR_BIT
#define BLOCKSIZE BYTESIZE
#define KEYSIZE 10
#define SUBKEYSIZE 8
#define SPLITKEYSIZE 5
UINT cbin2UINT(char*, UINT);
/*==========global variables==========*/
BYTE R1X=0;
BYTE R1Y=0;
BYTE C=0;
BYTE C2=0;
int r=0;
BYTE S0[]={1,0,2,3,3,1,0,2,2,0,3,1,1,3,2,0};
BYTE S1[]={0,1,2,3,2,0,1,3,3,0,1,0,2,1,0,3};
BYTE E[] = {0,2,1,3,0,1,2,3};
BYTE P4[] = {1,0,3,2};
ELEMENT DPS0[16][16];
ELEMENT DTS0[16][4];
ELEMENT DP2S0[16][16];
ELEMENT DT2S0[16][4];
BYTE R1XCHAR=0;
BYTE R1YCHAR=0;
int dex=0,dey=0,dex2=0,dey2=0;
double prob=0.0,prob2=0.0;
UINT key10 = 255;
```

```
/* ========== general functions ==========*/

void printBin(const char *str,unsigned int bInteger,unsigned int nSize){
char s[BYTESIZE*sizeof(UINT)];
UINT i;
UINT n=bInteger;
for(i=0; i<nSize; i++)
*(s + i)='0'; *(s+i)='\0';
i= nSize - 1;
while(n > 0){
s[i--]=(n % 2)? '1': '0';
n = n/2;
}

}
UINT cbin2UINT(char *s, unsigned int nSize){
int nLen = strlen(s);
UINT uResult = 0;
while (--nLen >= 0)
if(s[nLen] == '1')
uResult = 1 << (nSize - nLen - 1) | uResult;
return uResult;
}
/* ==========Key Scheduling ========== */
UINT leftShift(UINT nKey, UINT nShift, UINT nSize){
UINT n = nKey >> (nSize - nShift), i, nMask=0;
nKey <<= nShift;
for(i=0; i< nSize; i++)
nMask |= 1 << i;
return (nKey | n) & nMask;
}
```

```
UINT p10[] ={9,7,3,8,0,2,6,5,1,4};
UINT box_p10(UINT key10){
UINT uResult=0, i=0;
for(; i< KEYSIZE; i++)
if (1 << (KEYSIZE - p10[i] - 1) & key10)
uResult |= 1 << (KEYSIZE - i - 1);
return uResult;
}

void splitKey(UINT p10Key10, UINT uResult[]){
/**
* 31 == 0000011111
* 992 == 1111100000
*/
UINT H_SPLIT5BIT_MASK = 31, L_SPLIT5BIT_MASK = 992;
uResult[0] = (p10Key10 & L_SPLIT5BIT_MASK) >> SPLITKEYSIZE;
uResult[1] = p10Key10 & H_SPLIT5BIT_MASK;
}

UINT p8[] = {3,1,7,5,0,6,4,2};
UINT box_p8(UINT key5[]){
UINT uResult=0, uTemp, i=0;
UINT uMask = 255;
uTemp = key5[0] << SPLITKEYSIZE | key5[1];
uTemp &= uMask;
for(; i< SUBKEYSIZE; i++)
if (1 << (KEYSIZE - p8[i] - 1) & uTemp)
uResult |= 1 << (KEYSIZE - i - 3);
return uResult;
}
void keySchedule(UINT key10,UINT key8[]){
UINT key5[2]={0,0}, keyTemp, i;
```

```
keyTemp = box_p10(key10);
splitKey(keyTemp, key5);
for(i=0; i<2 ; i++){
key5[0] = leftShift(key5[0], i+1, SPLITKEYSIZE);
key5[1] = leftShift(key5[1], i+1, SPLITKEYSIZE);
key8[i]=box_p8(key5);
}
}
/* ==========IP and IP_1 ==========*/
UINT IP[] = {7,6,4,0,2,5,1,3};
UINT IP_1[]= {3,6,4,7,2,5,1,0};
BYTE per(UINT P[], BYTE input){
BYTE bRes = 00;
int i = 8;
while(--i >= 0)
if( 01 << (BLOCKSIZE - P[BLOCKSIZE - i - 1] - 1) & input )
bRes |= (01 << i);
return bRes;
}
/* ========== Round ==========
*/
void split824(BYTE bInput8, BYTE bLR[]){
BYTE L_mask = 240, H_mask = 15;
/** left */
bLR[0] = (bInput8 & L_mask) >> 4;
/** right */
bLR[1] = bInput8 & H_mask;
}
BYTE f(BYTE bRight, BYTE key){
BYTE bRes = 00, bTemp;
BYTE sLR4[]={0,0}, r, c;
int i = SUBKEYSIZE;
```

```
while(--i >= 0)
if( 01 << (4 - E[SUBKEYSIZE - i - 1] - 1) & bRight )
bRes |= (01 << i);
bRes ^= key;
split824(bRes,sLR4);
c = (sLR4[0] & 6) >> 1;
r = (sLR4[0] & 8) >> 2 | (sLR4[0] & 01);
sLR4[0] = S0[4*r + c] << 2;
c = (sLR4[1] & 6) >> 1;
r = (sLR4[1] & 8) >> 2 | (sLR4[1] & 01);
sLR4[1] = S1[4*r + c];
bTemp = sLR4[0] | sLR4[1];
bRes = 00;
// permute using P4
i=4;
while(--i >= 0)
if( 01 << (4 - P4[4 - i - 1] - 1) & bTemp )
bRes |= (01 << i);
return bRes;
}

int crypt(BYTE inputbits){
UINT key8[2]={0,0};
BYTE input8 = inputbits, exInput8, i;
/** left and Right */
BYTE LR[] ={00,00};
keySchedule(key10,key8);
// ===> Start of the round
exInput8 = input8;
R1X=exInput8;
for(i=0; i< 2; i++){
/** =====> begin round */
```

```
split824(exInput8,LR);
input8 = (f(LR[1],(BYTE)key8[i])^LR[0]) << 4;
input8 |= LR[1];
exInput8 = ((input8 & 240) >> 4) | ( (input8 & 15) << 4 );
/** ======> end of round */
if(i==0){
R1Y=exInput8;
}
}
if(r==0){
C=input8;
r++;
}else{
C2=input8;
r--;
}
return exInput8;
}
BYTE lastRound(BYTE inputbits,UINT key){
/** left and Right */
BYTE LR[] ={00,00};
BYTE input8 = inputbits;
BYTE exInput8;
exInput8 = input8;
split824(exInput8,LR);
input8 = (f(LR[1],(BYTE)key)^LR[0]) << 4;
input8 |= LR[1];
return input8;
}
/*=============================cryptanalytical functions=============================*/
void init(){
INDEX i,j;
```

```
for(i=0;i<16;i++)
for(j=0;j<16;j++){
DPS0[i][j]=0;
DP2S0[i][j]=0;
}
for(i=0;i<16;i++)
for(j=0;j<4;j++){
DTS0[i][j]=0;
DT2S0[i][j]=0;
}
}
ELEMENT SIO(COLUMN col,BYTE S[16]){
ELEMENT value;
BYTE r,c;
c = (col & 6) >> 1;
r = (col & 8) >> 2 | (col & 01);
value = S[4*r + c];
return value;
}
void difPair(){
COLUMN x=0;
COLUMN dx=0;
for(x=0;x<16;x++)
for(dx=0;dx<16;dx++){
DPS0[x][dx]=((SIO(x,S0))^(SIO(x^dx,S0)));
DP2S0[x][dx]=((SIO(x,S1))^(SIO(x^dx,S1)));
}
}
ELEMENT count(COLUMN dx,ROW dy,ELEMENT DS0[16][16]){
INDEX i;
int cnt=0;
for(i=0;i<16;i++)
```

```
if((DS0[i][dx])==dy)
cnt++;
return cnt;
}
void difTab(){
COLUMN dx=0;
ROW dy=0;
for(dx=0;dx<16;dx++){
for(dy=0;dy<4;dy++){
DTS0[dx][dy]=count(dx,dy,DPS0);
DT2S0[dx][dy]=count(dx,dy,DP2S0);
}
}
}
void printDPT(ELEMENT DS0[16][16]){
ROW x=0;
COLUMN dx=0;


void findDC(int ident,ELEMENT DTS[16][4]){
ELEMENT curV=0,curL=0;
INDEX i,j;
if(ident==0){
for(i=0;i<16;i++){
for(j=0;j<4;j++){
curV=DTS[i][j];
if((curV>curL)&(curV!=16)){
curL=curV;
dex=i;
dey=j;
}
}
```

```
}prob=((double)curL)/16;}else{
for(i=0;i<16;i++){
for(j=0;j<4;j++){
curV=DTS[i][j];
if((curV>curL)&(curV!=16)){
curL=curV;
dex2=i;
dey2=j;
}
}
} prob2=((double)curL)/16;
}
}
void printES(){
UINT key8[2]={0,0};
keySchedule(key10,key8);
printBin("Expected subkey = ",(key8[1]),SUBKEYSIZE);
}
void printGS(int K[256]){
ELEMENT curV=0,curL=0;
int i,k;
for(i=0;i<255;i++){
curV=K[i];
if(curV>curL){
curL=curV;
k=i;
}
}
printBin("Guessed Subkey = ",k,SUBKEYSIZE);
}
void extendDC(){
if((dex&8)==8){
```

```
R1XCHAR=(R1XCHAR|8);
R1YCHAR=(R1YCHAR|128);
}
if((dex&4)==4){
R1XCHAR=(R1XCHAR|2);
R1YCHAR=(R1YCHAR|32);
}
if((dex&2)==2){
R1XCHAR=(R1XCHAR|4);
R1YCHAR=(R1YCHAR|64);
}
if((dex&1)==1){
R1XCHAR=(R1XCHAR|1);
R1YCHAR=(R1YCHAR|16);
}
if((dex2&8)==8){
R1XCHAR=(R1XCHAR|8);
R1YCHAR=(R1YCHAR|128);
}
if((dex2&4)==4){
R1XCHAR=(R1XCHAR|4);
R1YCHAR=(R1YCHAR|64);
}
if((dex2&2)==2){
R1XCHAR=(R1XCHAR|2);
R1YCHAR=(R1YCHAR|32);
}
if((dex2&1)==1){
R1XCHAR=(R1XCHAR|1);
R1YCHAR=(R1YCHAR|16);
}
if((dey&2)==2)
```

```
R1YCHAR=(R1YCHAR|4);
if((dey&2)==1)
R1YCHAR=(R1YCHAR|8);
if((dey2&2)==2)
R1YCHAR=(R1YCHAR|1);
if((dey2&1)==1)
R1YCHAR=(R1YCHAR|2);
}
int main(void){
BYTE input=0;
BYTE dx=16;
BYTE curR1Y=0;
BYTE i;
BYTE k;
BYTE candidate=0;
BYTE lstest=0;
int count=0;
int PK2[256];
char cont,useKey;
char userKey[]="0000000000";
if(useKey!='y'){
key10 = cbin2UINT(userKey,KEYSIZE);
}
scanf("%c",&cont);
init();
difPair();
difTab();
printDPT(DPS0);
printDT(DTS0);
printDPT(DP2S0);
printDT(DT2S0);
findDC(0,DTS0);
```

```
findDC(1,DT2S0);
extendDC();
for(i=0;i<255;i++)
PK2[i]=0;
for(input=0;input<255;input++){
crypt(input);
curR1Y=R1Y;
crypt((BYTE)(input^R1XCHAR));
if((R1Y^curR1Y)==(R1YCHAR)){
count++;
for(k=0;k<255;k++){
if((lastRound(curR1Y,k)==C)&(lastRound(R1Y,k)==C2))
PK2[k]++;
}
}
}
for(i=0;i<255;i++)
printBin("Round 1 Input Characteristic = ",R1XCHAR,BLOCKSIZE);
printBin("Round 1 Output Characteristic = ",R1YCHAR,BLOCKSIZE);
return EXIT_SUCCESS;
}
```

Chapter 6
Testing

6. Testing

Once code has been generated, testing of the program begins. The testing process focuses on the logical internals of the software, ensuring that all statements have been tested, and on the functional externals; that is, conducting tests to uncover errors and ensure that defined input will produce actual results that agree with required results.

6.1 Objective of Testing

The overall objective of the testing process was to identify the maximum number of errors in the code with a minimum amount effort. Finding an error is thus considered a success rather than failure. On finding an error, efforts were made to correct it.

6.2 Inside the Testing Process

A test consists of a number of cases, where different aspects of the part under test are checked. While conducting the test, the deviations in the results are noted in a test protocol. Normally a deviation indicates an error in the code (although sometimes the test case is wrong and system is right). An error is noted and described in a test report for removal.

6.3 General Types of Errors

Errors can be of the following types:

- Functional (e.g., a function is missing or incorrect)
- Nonfunctional (e.g., performance is slow)
- Logical (e.g., user interface details are not considered logical)

6.4 Types of Testing

Given below are some of the different types of testing:

- Unit Testing
- Integration Testing
- System Testing
- Regression Testing

6.4.1 Unit Testing

A unit test is one of a component or a set of components, often conducted by the developer of the component.

6.4.2 Integration Testing

An integration testing is one of the packages that are put together where interface of packages and their collaborations are validated.

6.4.3 System Testing

System testing is functional testing of the entire system carried out by the end user. It is done on the basis of design of the system. It verifies that the system meets the specified functionality.

6.4.4 Regression Testing

It is basically a technique to handle changes in the system. A regression test is run after changes have been made to the system; it is actually a series of tests conducted on the entire system to determine whether any other functionality has been incorrectly affected by the changes. Continuous regression tests will unveil such problems.

6.5 Testing Cryptosys

Testing process of Cryptosys started as different units were completed. This project has been logically divided into three parts. Therefore, all the test phases were primarily implemented on each module, separately and collectively. Initially unit testing was performed on every program unit. Program units, syntax errors were removed. All scripting errors were detached and the validation checks were tested and corrected entirely. For semantic errors every program unit was tested with the help of test data.

Different program units were combined and the required functionality of the units were also tested. During the integration of the components, syntax and semantic errors were checked and removed on the client side.

All the modules were integrated after the completion of individual testing and again subjected to the testing phase. During this phase minor errors were encountered and removed.

Further more, the program itself checks for the validity of various inputs as shown in the figure below

Chapter 7

User Manual

7. User Manual

7.1 Executing Cryptosys

In order to execute the program it is required to first open the console environment and than execute the mpich as shown in the figure below

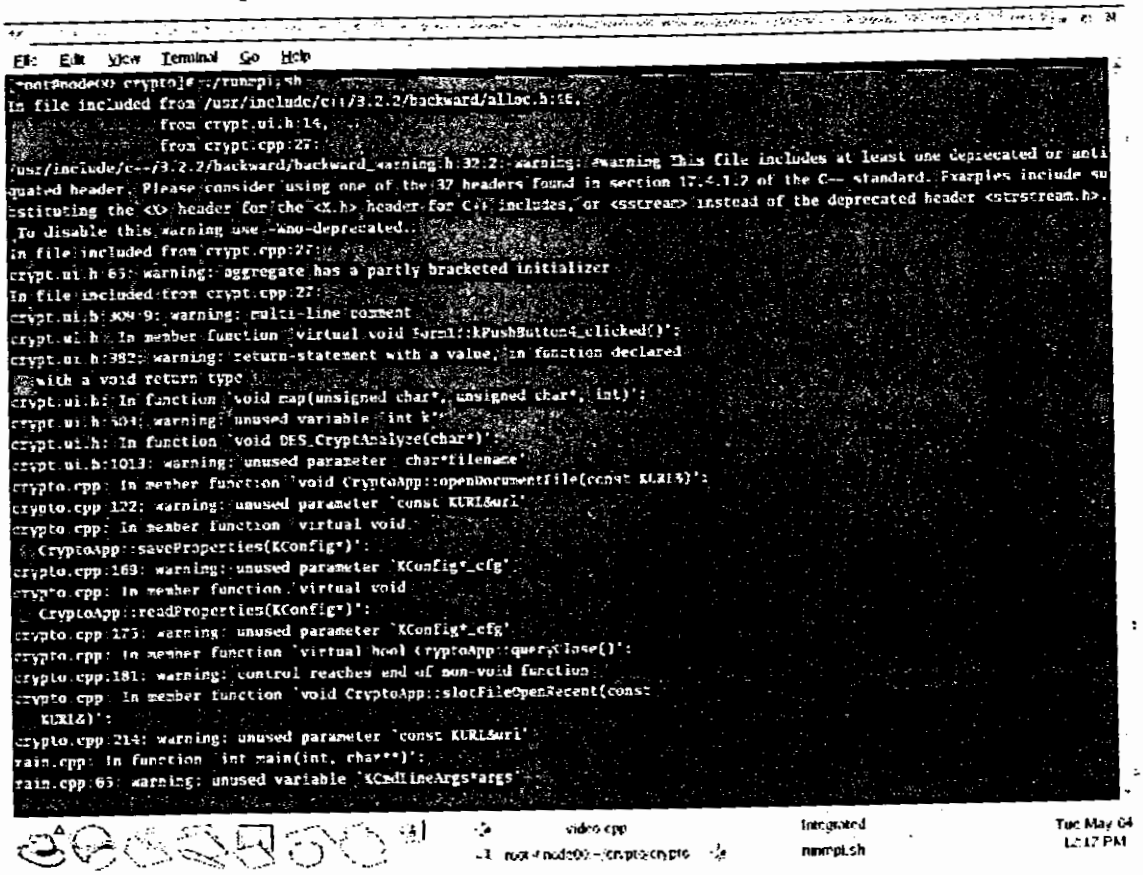


Figure 10 Executing Cryptosys

after executing the mpich the following dialogue box will appear here the user has to specify some input parameters.

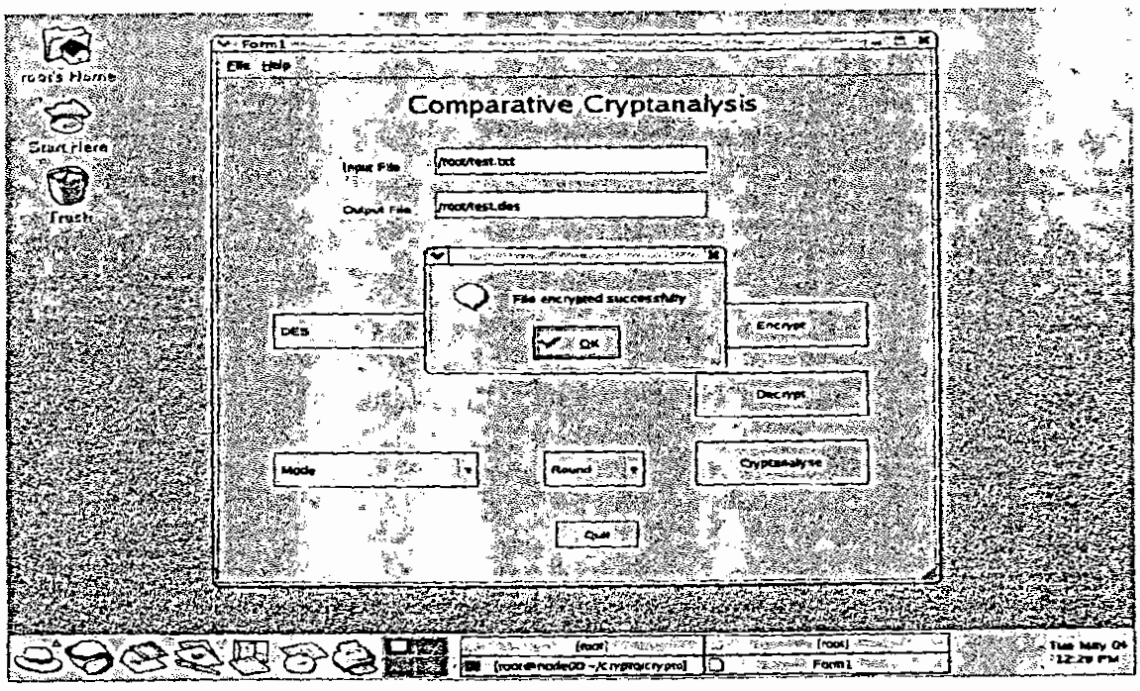


Figure 12 Encryption

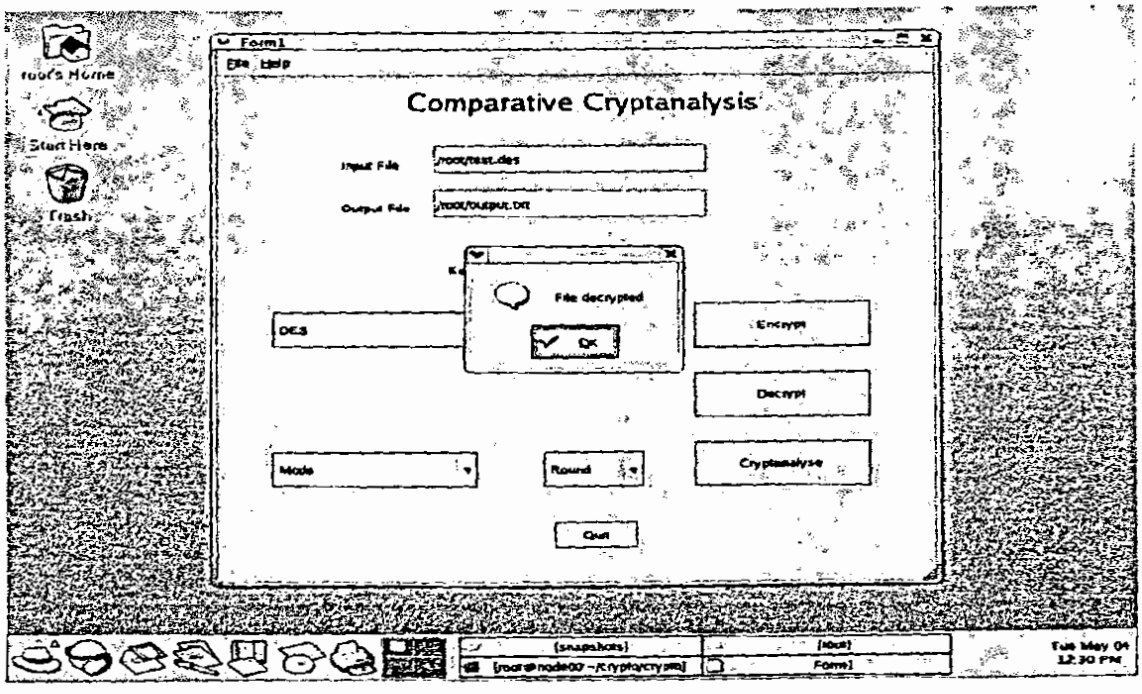


Figure 13 Decryption

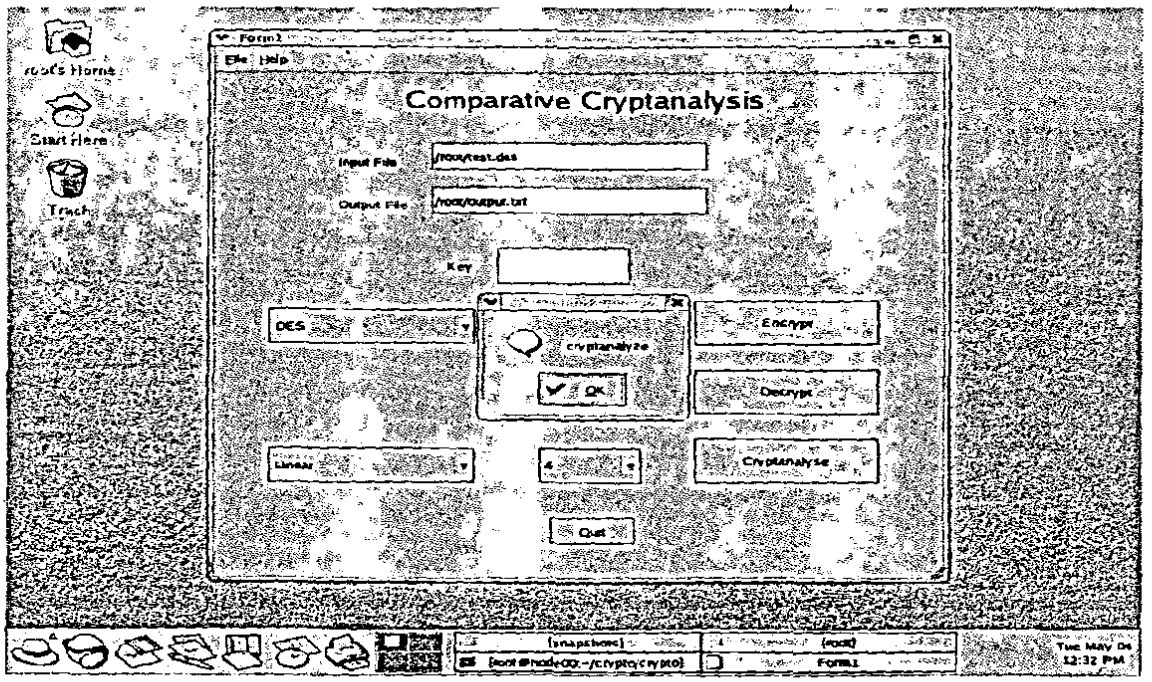


Figure 14 Cryptanalysis

after the attack is made on an encrypted file the new file contains the recovered text as shown in the figure below.

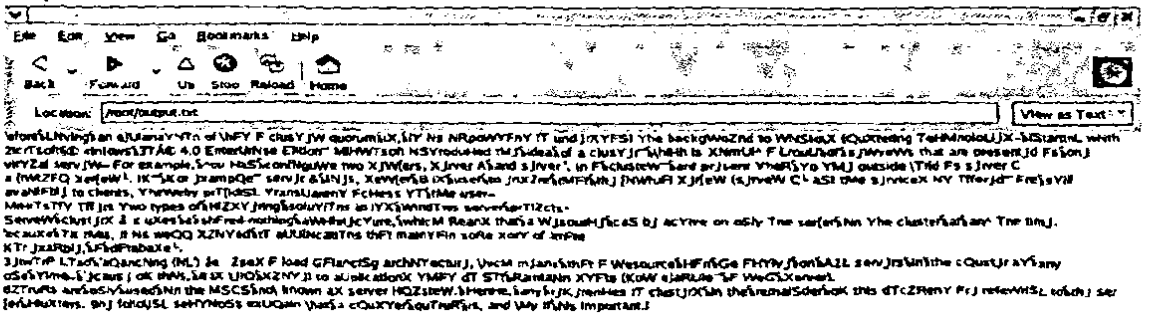


Figure 15 Output

The recovered text is not the same as the original plain text however, it recovers most part from the cipher text

Appendix

APPENDIX-A

A-1 Types of attacks on encrypted messages

Type of attacks	Known to Cryptanalyst
Cipher text only	Encryption algorithm Cipher text to be decoded
Known plain text	Encryption algorithm Cipher text to be decoded One or more plain text-cipher text pairs formed with the secret key .
Chosen plain text	Encryption algorithm Cipher text to be decoded Plain text message chosen by cryptanalyst, together with its corresponding cipher text generated with the secret key
Chosen cipher text	Encryption algorithm Cipher text to be decoded Cipher text message chosen by cryptanalyst, together with its corresponding plain text generated with the secret key
Chosen text	Encryption algorithm Cipher text to be decoded Plain text message chosen by cryptanalyst, together with its corresponding cipher text generated with the secret key Cipher text message chosen by cryptanalyst, together with its corresponding plain text generated with the secret key

A-2 Time Required for Exhaustive Key Search

Key Size	Number of alternatives	One Encryption/μs	10^6Encryptions/μs
32 bits	$2^{32}=4.3*10^9$	2^{31}μs=35.8minutes	2.15ms
56 bits	$2^{56}=7.2*10^{16}$	2^{55}μs=1142 years	10.01h
128 bits	$2^{128}=3.4*10^{38}$	2^{127}μs=$5.4*10^{24}$ years	$5.4*10^{18}$ years
26 characters	$26!=4.03*10^{26}$	$2*10^{26}$μs=$6.4*10^{12}$years	$6.4*10^6$years

APPENDIX-B

B-1 Clustering Methods

Clustering Methods	Description	Benefits	Limitations
Passive Standby	A passive server takes over in case of primary server failure.	Easy to implement.	High cost because the secondary server is un available for other processing tasks.
Active Secondary	The secondary server is also used for processing tasks.	Reduced cost because the secondary server is also available for processing.	Increased complexity.
Separate Servers	Separate servers have their own disks. Data is continuously copied from primary to secondary server.	High availability.	High network and server overhead due to copying operations.
Servers Connected to Disks	Servers are cabled to the same disks, but each server owns its disks. If one disks are taken over by other server.	Reduced network and server overhead due to elimination of copying operations.	Usually required disk mirroring or RAID technology to compensate for risk of disk failure.
Servers Share Disks	Multiple servers simultaneously share access to disks.	Low network and server overhead. Reduced risk of downtime caused by disk failure.	Requires lock manager software. Usually used with disk mirroring or RAID technology.

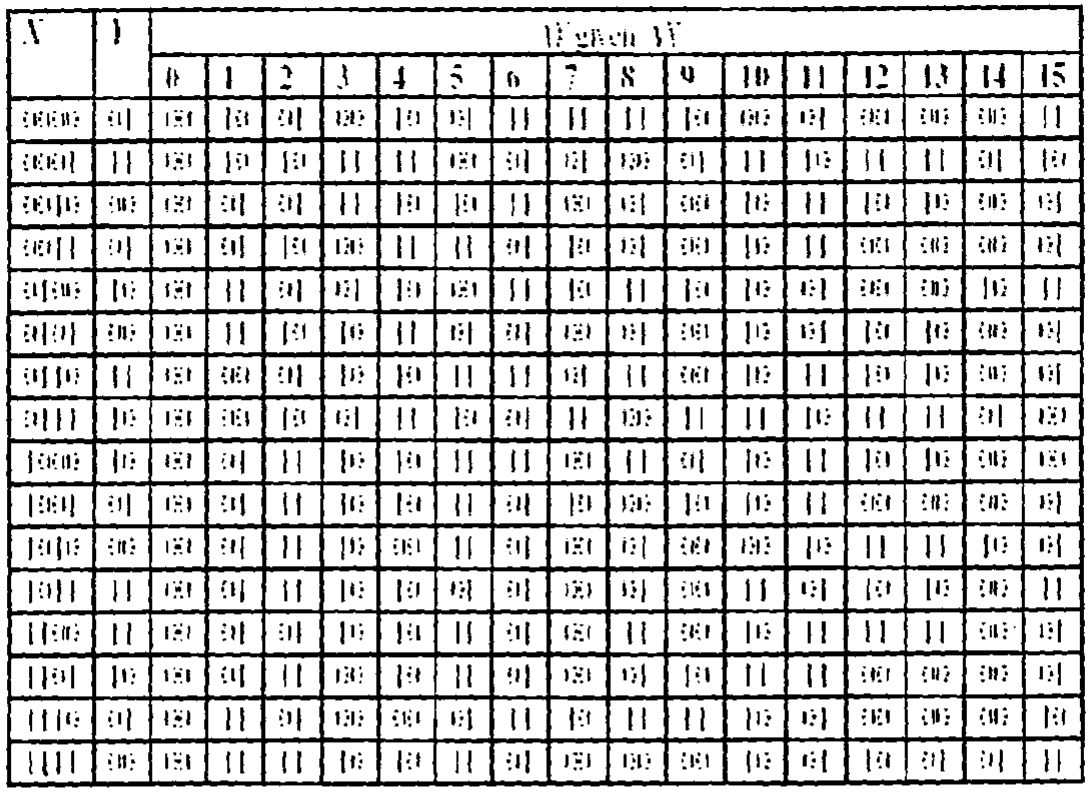
APPENDIX-C

C-1 Difference Pair of S_0

ΔX	ΔY	W given W															
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0000	01	00	10	01	00	11	01	10	11	11	00	01	10	10	11	00	01
0001	11	00	10	10	11	11	01	01	00	10	01	00	11	01	00	11	10
0010	00	00	01	01	11	11	10	10	00	00	11	10	01	01	00	11	10
0011	01	00	01	10	00	11	10	01	11	10	01	00	11	01	00	11	10
0100	10	00	10	01	00	11	01	10	11	01	00	11	10	00	11	10	01
0101	00	00	10	10	11	11	01	01	00	10	11	00	01	01	10	11	00
0110	11	00	01	01	11	11	10	10	00	10	11	00	01	11	00	01	10
0111	10	00	01	10	00	11	10	01	11	10	11	00	01	01	10	11	00
1000	10	00	11	10	01	01	00	11	10	11	01	10	11	00	10	01	00
1001	01	00	11	10	01	11	10	01	00	10	00	00	01	01	11	11	10
1010	00	00	11	10	01	01	00	11	10	00	01	01	11	11	10	10	00
1011	11	00	11	10	01	11	10	01	00	10	11	00	10	01	00	11	01
1100	11	00	01	10	11	01	10	11	00	01	11	00	01	10	00	11	10
1101	10	00	01	10	11	11	00	01	10	10	00	00	01	01	11	11	10
1110	01	00	01	10	11	01	10	11	00	10	11	11	01	01	00	00	10
1111	00	00	01	10	11	11	00	01	10	10	11	00	10	01	00	11	01

Where ΔX is Hexadecimal

C-2 Difference Pair of S_1



Where ΔX is Hexadecimal

C-3 Difference Distribution Table for S0

Input Difference ΔX	Output Difference ΔY			
	0	1	2	3
0	16	0	0	0
1	0	8	4	4
2	0	4	12	0
3	4	4	0	8
4	0	4	0	12
5	4	4	8	0
6	0	8	4	4
7	8	0	4	4
8	4	2	10	2
9	4	4	0	8

10	10	2	2	2
11	0	8	4	4
12	2	10	2	2
13	8	0	4	4
14	2	2	2	10
15	4	4	8	0

Where ΔX ΔY are in Hexadecimal

C-4 Difference Distribution Table for S_1

Input Difference ΔY	Output Difference ΔY			
	0	1	2	3
0	16	0	0	0
1	2	8	2	4
2	0	6	4	6
3	4	2	8	2
4	2	0	10	4
5	2	4	2	8
6	0	1	0	6
7	8	2	4	2
8	4	6	0	6
9	8	2	4	2
10	2	0	10	4
11	0	6	4	6
12	0	6	4	6
13	6	0	6	4
14	1	4	2	0
15	2	8	2	4

Where ΔX ΔY are in Hexadecimal

C-5 I/O Table for S_0

X_3	X_2	X_1	X_0	Y_1	Y_0
0	0	0	0	0	1
0	0	0	1	1	1
0	0	1	0	0	0
0	0	1	1	0	1
0	1	0	0	1	0
0	1	0	1	0	0
0	1	1	0	1	1
0	1	1	1	1	0
1	0	0	0	1	0
1	0	0	1	0	1
1	0	1	0	0	0
1	0	1	1	1	1
1	1	0	0	1	1
1	1	0	1	1	0
1	1	1	0	0	1
1	1	1	1	0	0

C-6 Distribution Table for S_0

α	β		
	1	2	3
1	0	0	0
2	0	0	-2
3	0	-2	-2
4	-2	2	0
5	0	2	0
6	-2	0	-2
7	-2	0	-2
8	0	0	0
9	0	0	0
10	0	2	2
11	0	2	-4
12	-2	2	0
13	-2	2	0
14	-2	-4	2
15	-2	4	2

APPENDIX-D

Activity Diagram

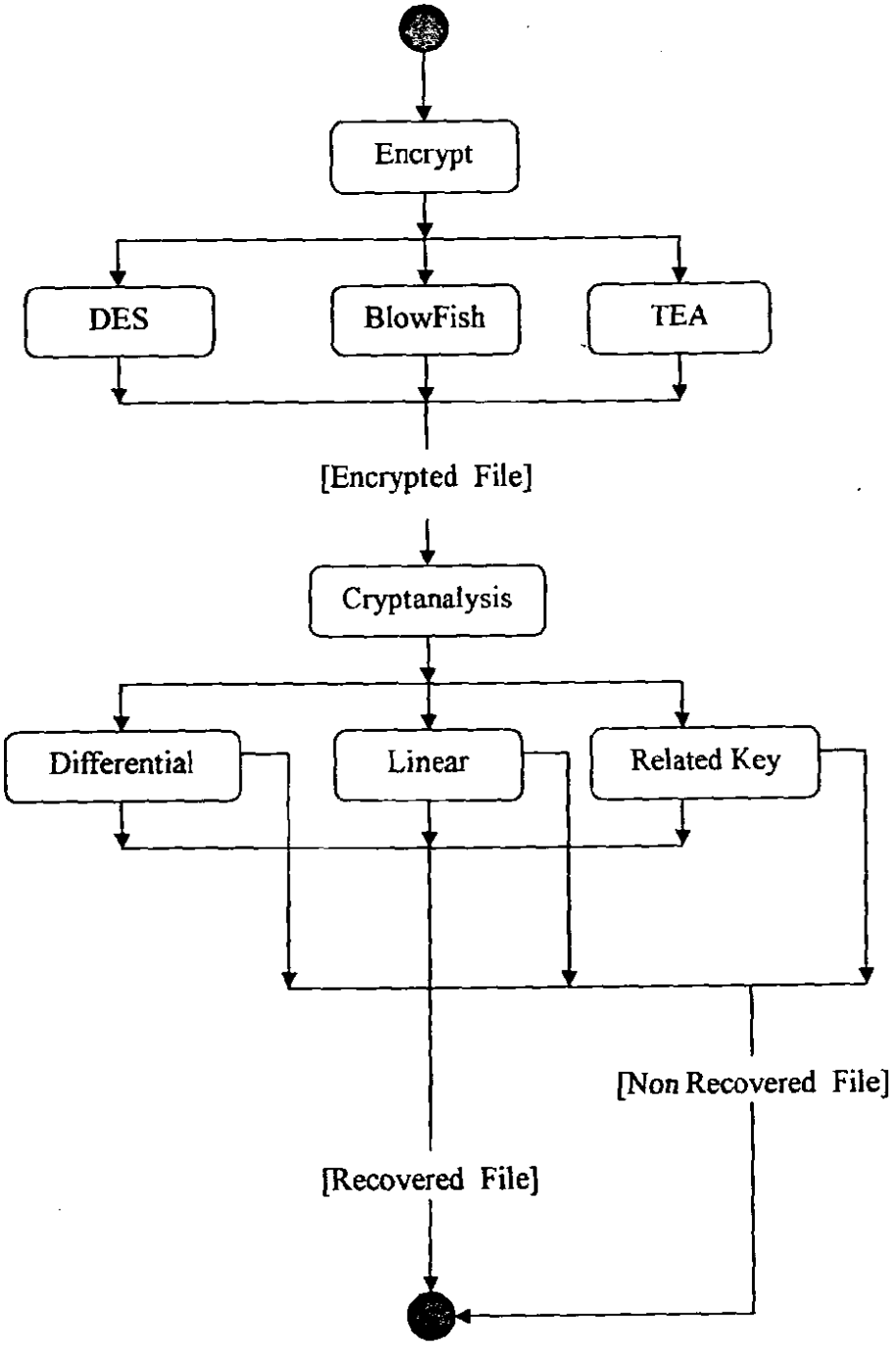


Figure 16 Activity Diagram

Data Flow Diagram

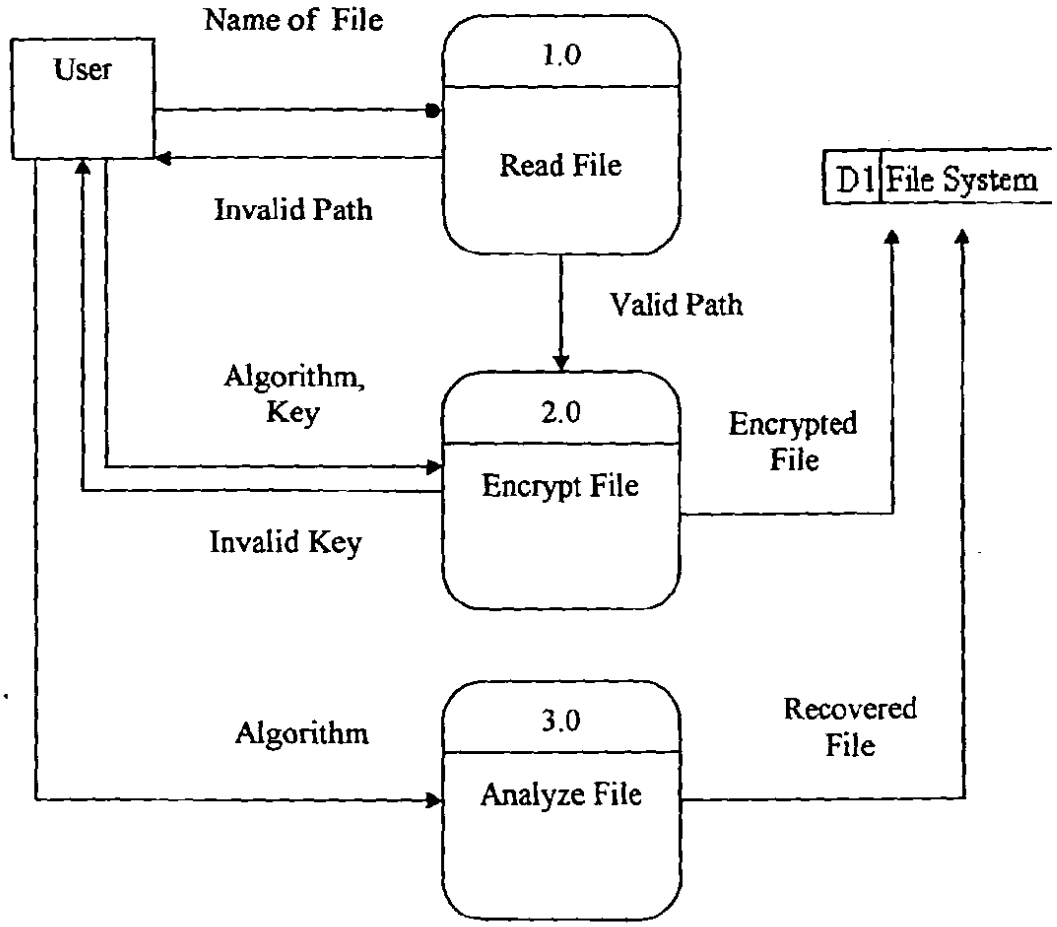


Figure 17 Data Flow Diagram

APPENDIX-E

Data Encryption Standard (DES)

The Data Encryption Standard (DES) consists of the following Data Encryption Algorithm to be implemented in special purpose electronic devices. These devices shall be designed in such a way that they may be used in a computer system or network to provide cryptographic protection to binary coded data. The method of implementation will depend on the application and environment. The devices shall be implemented in such a way that they may be tested and validated as accurately performing the transformations specified in the following algorithm

Data Encryption Algorithm

The algorithm is designed to encipher and decipher blocks of data consisting of 64 bits under control of a 64-bit key. Deciphering must be accomplished by using the same key as for enciphering, but with the schedule of addressing the key bits altered so that the deciphering process is the reverse of the enciphering process. A block to be enciphered is subjected to an initial permutation IP, then to a complex key-dependent computation and finally to a permutation which is the inverse of the initial permutation IP-1. The key-dependent computation can be simply defined in terms of a function f, called the cipher function, and a function KS, called the key schedule. A description of the computation is given first, along with details as to how the algorithm is used for encipherment. Next, the use of the algorithm for decipherment is described. Finally, a definition of the cipher function f is given in terms of primitive functions which are called the selection functions Si and the permutation function P.

The following notation is convenient: Given two blocks L and R of bits, LR denotes the block consisting of the bits of L followed by the bits of R. Since concatenation is associative, B1B2...B8, for example, denotes the block consisting of the bits of B1 followed by the bits of B2...followed by the bits of B8. Blocks are composed of bits numbered from left to right, i.e., the left most bit of a block is bit one.

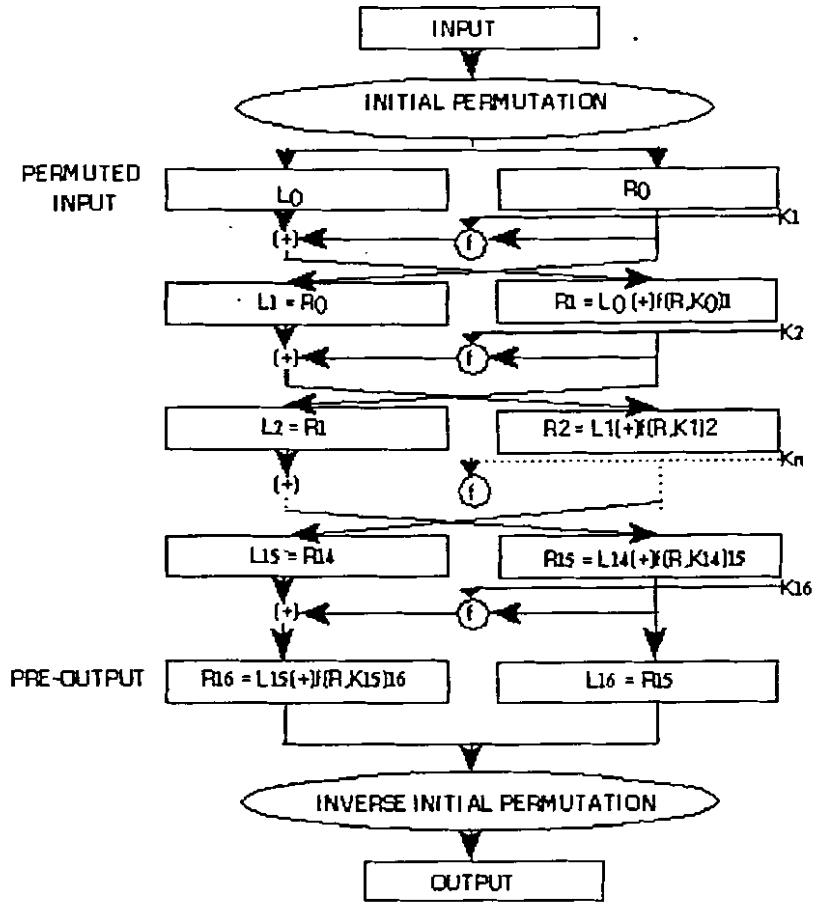


Figure 18 Enciphering Computation

Enciphering

A sketch of the enciphering computation is given in Figure A.1. The 64 bits of the input block to be enciphered are first subjected to the following permutation, called the initial permutation IP:

IP

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

That is the permuted input has bit 58 of the input as its first bit, bit 50 as its second bit, and so on with bit 7 as its last bit. The permuted input block is then the input to a complex key-dependent computation described below. The output of that computation, called the preoutput, is then subjected to the following permutation which is the inverse of the initial permutation:

IP-1

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

That is, the output of the algorithm has bit 40 of the preoutput block as its first bit, bit 8 as its second bit, and so on, until bit 25 of the preoutput block is the last bit of the output.

The computation which uses the permuted input block as its input to produce the preoutput block consists, but for a final interchange of blocks, of 16 iterations of a calculation that is described below in terms of the cipher function f which operates on two blocks, one of 32 bits and one of 48 bits, and produces a block of 32 bits.

Let the 64 bits of the input block to an iteration consist of a 32 bit block L followed by a 32 bit block R. Using the notation defined in the introduction, the input block is then LR.

Let K be a block of 48 bits chosen from the 64-bit key. Then the output L'R' of an iteration with input LR is defined by:

1. $L' = R$

   $R' = L(+)f(R,K)$

where (+) denotes bit-by-bit addition modulo 2.

As remarked before, the input of the first iteration of the calculation is the permuted input block. If L'R' is the output of the 16th iteration then R'L' is the preoutput block. At each iteration a different block K of key bits is chosen from the 64-bit key designated by KEY.

With more notations we can describe the iterations of the computation in more detail. Let KS be a function which takes an integer n in the range from 1 to 16 and a 64-bit block KEY as input and yields as output a 48-bit block Kn which is a permuted selection of bits from KEY. That is

2. Kn = KS(n,KEY)

with Kn determined by the bits in 48 distinct bit positions of KEY. KS is called the key schedule because the block K used in the n'th iteration of (1) is the block Kn determined by (2).

As before, let the permuted input block be LR. Finally, let L() and R() be respectively L and R and let Ln and Rn be respectively L' and R' of (1) when L and R are respectively Ln-1 and Rn-1 and K is Kn; that is, when n is in the range from 1 to 16,

3. Ln = Rn-1
   nn = Ln-1(+)f(Rn-1,Kn)

The preoutput block is then R16L16.
The key schedule KS of the algorithm is described in detail in the Appendix. The key schedule produces the 16 Kn which are required for the algorithm.

Deciphering

The permutation IP-1 applied to the preoutput block is the inverse of the initial permutation IP applied to the input. Further, from (1) it follows that:

4. R = L'
   L = R' (+) f(L',K)

Consequently, to decipher it is only necessary to apply the very same algorithm to an enciphered message block, taking care that at each iteration of the computation the same block of key bits K is used during decipherment as was used during the encipherment of the block. Using the notation of the previous section, this can be expressed by the equations:

5. $Rn\text{-}1 = Ln$

$Ln\text{-}1 = Rn\ (+)\ f(Ln,Kn)$

where now R16L16 is the permuted input block for the deciphering calculation and L() and R() is the preoutput block. That is, for the decipherment calculation with R16L16 as the permuted input, K16 is used in the first iteration, K15 in the second, and so on, with K1 used in the 16th iteration.

The Cipher Function f, A sketch of the calculation of f(R,K) is given in Figure A.2

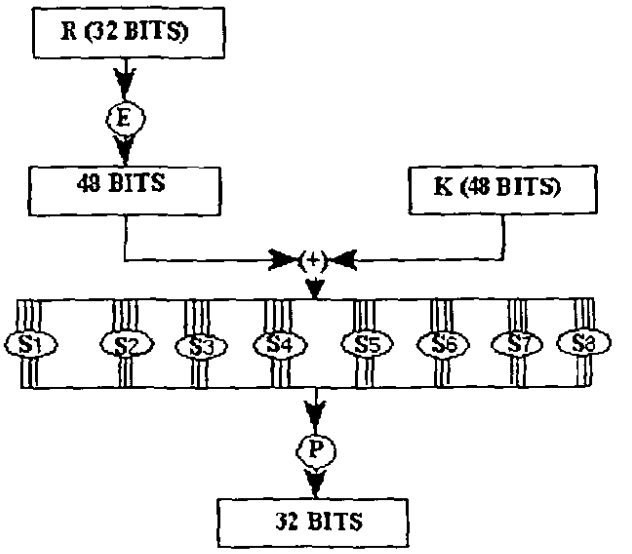


Figure 19 Calculation of f(R,K)

Let E denote a function which takes a block of 32 bits as input and yields a block of 48 bits as output. Let E be such that the 48 bits of its output, written as 8 blocks of 6 bits each, are obtained by selecting the bits in its inputs in order according to the following table:

E Bit-Selection Table

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

Thus the first three bits of E(R) are the bits in positions 32, 1 and 2 of R while the last 2 bits of E(R) are the bits in positions 32 and 1.

Each of the unique selection functions $S_1,S_2,...,S_8$, takes a 6-bit block as input and yields a 4-bit block as output and is illustrated by using a table containing the recommended S_1:

S_1

14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

If S_1 is the function defined in this table and B is a block of 6 bits, then $S_1(B)$is determined as follows: The first and last bits of B represent in base 2 a number in the range 0 to 3. Let that number be i. The middle 4 bits of B represent in base 2 a number in the range 0 to 15. Let that number be j. Look up in the table the number in the i'th row and j'th column. It is a number in the range 0 to 15 and is uniquely represented by a 4 bit block. That block is the output $S_1(B)$ of S_1 for the input B. For example, for input 011011 the row is 01, that is row 1, and the column is determined by 1101, that is column 13. In row 1 column 13 appears 5 so that the output is 0101.

The permutation function P yields a 32-bit output from a 32-bit input by permuting the bits of the input block. Such a function is defined by the following table:

P

16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

The output P(L) for the function P defined by this table is obtained from the input L by taking the 16th bit of L as the first bit of P(L), the 7th bit as the second bit of P(L), and so on until the 25th bit of L is taken as the 32nd bit of P(L).

Now let $S_1,...,S_8$ be eight distinct selection functions, let P be the permutation function and let E be the function defined above.

To define f(R,K) we first define $B_1,...,B_8$ to be blocks of 6 bits each for which

6 $B_1B_2...B_8 = K(+)E(R)$

The block f(R,K) is then defined to be

7 $P(S_1(B_1)S_2(B_2)...S_8(B_8))$

Thus K(+)E(R) is first divided into the 8 blocks as indicated in (6). Then each Bi is taken as an input to S_i and the 8 blocks $(S_1(B_1)S_2(B_2)...S_8(B_8)$ of 4 bits each are consolidated into a single block of 32 bits which forms the input to P. The output (7) is then the output of the function f for the inputs R and K.

Primitive Functions for the Data Encryption Algorithm

The choice of the primitive functions KS, $S_1,...,S_8$ and P is critical to the strength of an encipherment resulting from the algorithm. Specified below is the recommended set of

functions, describing $S_1,...,S_8$ and P in the same way they are described in the algorithm. For the interpretation of the tables describing these functions, see the discussion in the body of the algorithm.

The primitive functions $S_1,...,S_8$ are:

S_1

14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S_2

15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

S_3

10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

S_4

7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

S_5

2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6

4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

S_6

12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

S_7

4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

S_8

13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

The primitive function **P** is:

16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

Recall that K_n, for $1 <= n <= 16$, is the block of 48 bits in (2) of the algorithm. Hence, to describe K_S, it is sufficient to describe the calculation of K_n from KEY for n = 1, 2,..., 16. That calculation is illustrated in Figure A.3. To complete the definition of K_S it is therefore sufficient to describe the two permuted choices, as well as the schedule of left shifts. One bit in each 8-bit byte of the KEY may be utilized for error detection in key generation, distribution and storage. Bits 8, 16,..., 64 are for use in assuring that each byte is of odd parity.

Permuted choice 1 is determined by the following table

PC-1

57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36

63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

The table has been divided into two parts, with the first part determining how the bits of C_0 are chosen, and the second part determining how the bits of D_0 are chosen. The bits of KEY are numbered 1 through 64. The bits of C_0 are respectively bits 57, 49, 41,..., 44 and 36 of KEY, with the bits of D_0 being bits 63, 55, 47,..., 12 and 4 of KEY.

With C_0 and D_0 defined, we now define how the blocks C_n and D_n are obtained from the blocks C_{n-1} and D_{n-1}, respectively, for n = 1, 2,..., 16. That is accomplished by adhering to the following schedule of left shifts of the individual blocks:

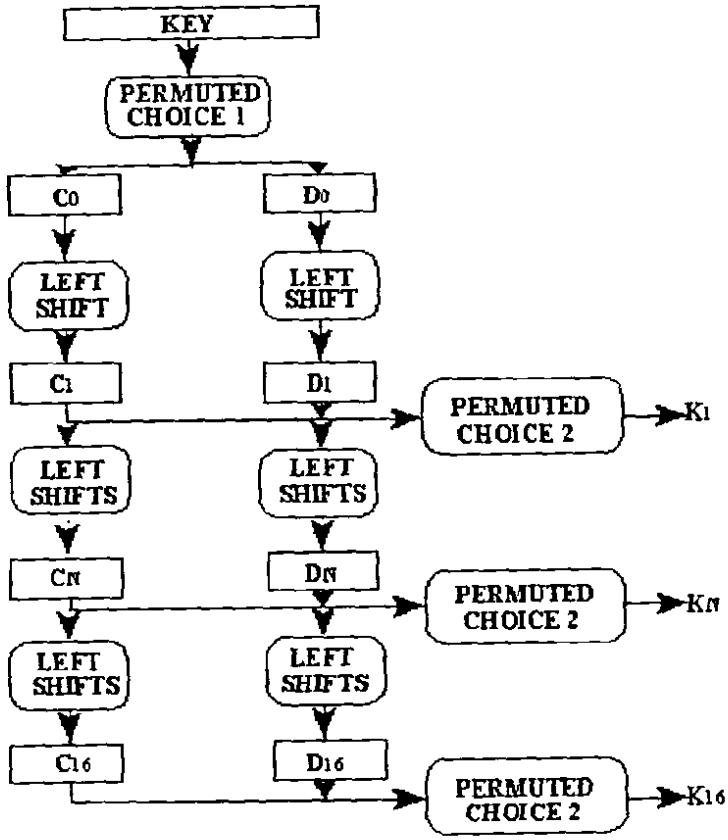


Figure 20 Key Schedule Calculation

Iteration Number	Number of Left Shifts
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2
9	1
10	2

11	2
12	2
13	2
14	2
15	2
16	1

For example, C_3 and D_3 are obtained from C_2 and D_2, respectively, by two left shifts, and C_{16} and D_{16} are obtained from C_{15} and D_{15}, respectively, by one left shift. In all cases, by a single left shift is meant a rotation of the bits one place to the left, so that after one left shift the bits in the 28 positions are the bits that were previously in positions 2, 3,..., 28, 1.

Permuted choice 2 is determined by the following table:

PC-2

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2

41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

Therefore, the first bit of K_n is the 14th bit of C_nD_n, the second bit the 17th, and so on with the 47th bit the 29th, and the 48th bit the 32nd.

Tiny Encryption Algorithm

The Tiny Encryption Algorithm (TEA) and related variants (XTEA, Block TEA, XXTEA) are block ciphers notable for their simplicity of description and implementation (typically a few lines of code), and consequently enjoy a measure of popularity.

TEA [WheelerNeedham94] operates on 64-bit message blocks with a 128-bit key, and is a Feistel network with a suggested 64 rounds (though the authors speculate that 32 rounds might suffice).

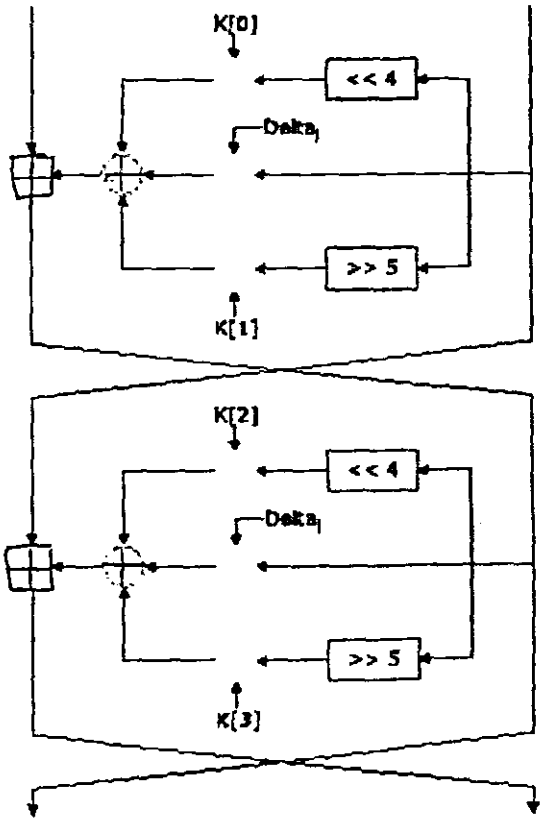


Figure 21 Tiny Encryption Algorithm

Two TEA Feistel rounds are termed a cycle. One cycle of TEA is illustrated in Figure A.4 The algorithm uses multiples of a magic constant,δ, derived from the golden ratio, to ensure that the encryption in each cycle is different; the precise value of δ s probably unimportant, but for TEA it is defined as:

$$\delta = \lfloor (\sqrt{5} - 1)2^{31} \rfloor.$$

Blowfish Algorithm

Blowfish is a variable-length key block cipher. It is only suitable for applications where the key does not change often, like a communications link or an automatic file encryptor. It is significantly faster than DES when implemented on 32-bit microprocessors with large data caches, such as the Pentium and the PowerPC.

Description of the Algorithm

Blowfish is a variable-length key, 64-bit block cipher. The algorithm consists of two parts: a key-expansion part and a data- encryption part. Key expansion converts a key of at most 448 bits into several subkey arrays totaling 4168 bytes.

Data encryption occurs via a 16-round Feistel network. Each round consists of a key-dependent permutation, and a key- and data-dependent substitution. All operations are XORs and additions on 32-bit words. The only additional operations are four indexed array data lookups per round.

Subkeys

Blowfish uses a large number of subkeys. These keys must be precomputed before any data encryption or decryption.

1. The P-array consists of 18 32-bit subkeys:
   P1, P2,..., P18.
2. There are four 32-bit S-boxes with 256 entries each:
   S1,0, S1,1,..., S1,255;
   S2,0, S2,1,.., S2,255;
   S3,0, S3,1,..., S3,255;
   S4,0, S4,1,.., S4,255.

The exact method used to calculate these subkeys will be described later.

Encryption

Blowfish is a Feistel network consisting of 16 rounds (see Figure 1). The input is a 64-bit data element, x.

Divide x into two 32-bit halves: xL, xR

For i = 1 to 16:

xL = xL ⊕ Pi

xR = F(xL) ⊕ xR

Swap xL and xR

Next i

Swap xL and xR (Undo the last swap.)

xR = xR ⊕ P17

xL = xL ⊕ P18

Recombine xL and xR

Function F

Divide xL into four eight-bit quarters: a, b, c, and d

F(xL) = ((S1,a + S2,b mod 232) ⊕ S3,c) + S4,d mod 232

Decryption

Decryption is exactly the same as encryption, except that P1, P2,..., P18 are used in the reverse order. Implementations of Blowfish that require the fastest speeds should unroll the loop and ensure that all subkeys are stored in cache.

Generating the Subkeys

The subkeys are calculated using the Blowfish algorithm. The exact method is as follows:

1. Initialize first the P-array and then the four S-boxes, in order, with a fixed string. This string consists of the hexadecimal digits of pi (less the initial 3). For example:

   P1 = 0x243f6a88

   P2 = 0x85a308d3

   P3 = 0x13198a2e

   P4 = 0x03707344

2. XOR P1 with the first 32 bits of the key, XOR P2 with the second 32-bits of the key, and so on for all bits of the key (possibly up to P14). Repeatedly cycle through the key bits until the entire P-array has been XORed with key bits. (For every short key, there is at least one equivalent longer key; for example, if A is a 64-bit key, then AA, AAA, etc., are equivalent keys.)

3. Encrypt the all-zero string with the Blowfish algorithm, using the subkeys described in steps (1) and (2).
4. Replace P1 and P2 with the output of step (3).
5. Encrypt the output of step (3) using the Blowfish algorithm with the modified subkeys.
6. Replace P3 and P4 with the output of step (5).
7. Continue the process, replacing all entries of the P- array, and then all four S-boxes in order, with the output of the continuously-changing Blowfish algorithm.

In total, 521 iterations are required to generate all required subkeys. Applications can store the subkeys rather than execute this derivation process multiple times.

Glossary

Glossary

Iterative:

A system that takes a fairly simple encryption rule and applies it many times to obtain data security.

Plaintext:

The message before going through the encryption, usually abbreviated by the symbol P.

Cipher:

Some method to change a message into a non readable format letter by letter.

Encrypt:

To change a message into the unreadable form that can be openly transmitted.

Decrypt:

To change a message from the unreadable form that it was transmitted in to the readable form for human or machine reading. The opposite of encrypt.

Secure:

Effectively unbreakable. A message is considered secure if by the time it can be decrypted by an unintended receiver, the information is useless. Note that for some kinds of information, that means that the message must be effectively impossible to decrypt by an unintended receiver.

Cryptanalysis:

The art and science of decoding an encrypted message without being the intended receiver, and work which aids that end.

Key:

A short piece of information that is used with the rule for decryption to allow an authorized user to decrypt a message. Often abbreviated as K.

Private Key:

An encryption system that uses a single key for both encryption and decryption, thus the key must be kept private.

Public Key:

An encryption system that uses two keys, so one key is kept private, and the other key can be publicly distributed. One key can be used to encrypt, but the other key is required to decrypt.

DES:

Data Encryption Standard. A standard put forth by the US government around 1977 for sensitive but non-classified information. Uses a 56 bit key (64 plus 8 parity bits), and 16 rounds. Still considered a reasonably good standard. Hard to implement in software, easy in hardware. Developed by IBM with technical assistance from the US government. In 1993 it was renewed for the standard for another 20 years. Some methods now exist that make the most primitive use of this system slightly less secure, and hypothetically breakable given an inordinate amount of time and resources.

Node:

A node can be a single or multiprocessor system (PCs, workstations or SMPs) with memory, I/O facilities, and an operating system.

SAC:

Strict Avalanche Criterion

GAC:

Global Avalanche Characteristic

Absolute Scalability:

It is possible to create large clusters that far surpass the power of even the largest stand alone machine.

Incremental Scalability:

A cluster is configured in such a way that it is possible to add new systems to the cluster.

High Availability:

Each node in a cluster is a standalone therefore a failure to a node need not affect the other nodes.

Superior Price/Performance:

It is possible to put together a cluster with equal or greater computing power then a single large machine.

References

References

[Adams92] Adams, C. 1992. On immunity against Biham and Shamir's differential cryptanalysis. Information Processing Letters. 41: 77-80.

[Ayoub82] Ayoub, F. 1982. Probabilistic completeness of substitution-permutation encryption networks. IEE Proceedings E. 129(5): 195-199.

[BellareKohno03] M Bellare, T Kohno "A theoretical treatement of Related key attacks RKA-PRps, RKA-PRFs and Applications" Advances in cryptology-EUROCRYPT 2003

[Biham94] E Biham. New Type of Cryptanalytic Attacks Using Related Key. Advances in Cryptology- EUROCRYPT 94.

[BihamShamir90] Biham, E. and A. Shamir. 1990. Differential Cryptanalysis of DES-like Cryptosystems. Advances in Cryptology -- CRYPTO '90. Springer-Verlag. 2-21.

[BihamShamir91] Biham, E. and A. Shamir. 1991. Differential Cryptanalysis of Snefru, Khafre, REDOC-II, LOKI and Lucifer. Advances in Cryptology -- CRYPTO '91. Springer-Verlag. 156-171.

[BihamShamir92] Biham, E. and A. Shamir. 1992. Differential Cryptanalysis of the Full 16-round DES. Advances in Cryptology -- CRYPTO '92. Springer-Verlag. 487-496

[BS93] Bruce Schneier Applied Cryptography Porotocols and Source Code in C, John Wiley and Sons Inc.

[ButtyanVajda95] Buttyan, L. and I Vajda. 1995. Searching for the best linear approximation of DES-like cryptosystems. Electronics Letters. 31(11): 873-874.

[CBT] http://www.ciphersbyritter.com/

[CietPiretJean03] Mathieu Ciet, Gilles Piret and Jean Jacques Quisquater

"Related key and Slide attacks Analysis Connections and Improvements"

[Connor93] O'Connor, L. 1993. On the Distribution of Characteristics in Bijective Mappings. Advances in Cryptology -- EUROCRYPT 93. 360-370.

[Cusick93] Cusick, T. 1993. Boolean functions satisfying a higher order strict avalanche criterion. Advances in Cryptology -- EUROCRYPT '93. 102-117.

[DaeGovVande94] Daemen, J., R. Govaerts and J. Vandewalle. 1994. Correlation Matrices. Fast Software Encryption. Lecture Notes in Computer Science (LNCS) 1008. Springer-Verlag. 275-285.

[DawsonTavares91] Dawson, M. and S. Tavares. 1991. An Expanded Set of S-box Design Criteria Based on Information Theory and its Relation to Differential-Like Attacks. Advances in Cryptology -- EUROCRYPT '91. 353-367.

[DiffleHellman79] W. Diffle and M.E. Hellman. \Privacy and Authentication: An Introduction to Cryptography". Proceedings of the IEEE, vol 67 no 3, March 1979.

[Fauza00] Fauza Mirzan (2000),Block Ciphers and Cryptanalysis, Department of Mathematics, Royal Holloway University of London

[Feistel73] Feistel, H. 1973. Cryptography and Computer Privacy. Scientific American. 228(5): 15-23.

[Forre88] Forre, R. 1988. The Strict Avalanche Criterion: Spectral Properties of Boolean Functions and an Extended Definition. Advances in Cryptology -- CRYPTO '88. 450-468.

[GordonRetkin82] Gordon, J. and H. Retkin. 1982. Are Big S-Boxes Best?

[HarpeKraMassey95] Harpes, C., G. Kramer and J. Massey. 1995. A

Generalization of Linear Cryptanalysis and the Applicability of Matsui's Piling-up Lemma. Advances in Cryptology -- EUROCRYPT '95. 24-38.

[Howard00] Howard M. Heyz (2000), A Tutorial on Linear and Differential Cryptanalysis, Memorial University of Newfoundland, Canada

[JohnBruceDavid96] J Kelsey, B Schneier, D Wanger "Key Schedule Cryptanalysis of IDEA, G-DES, GOST, SAFER and Triple-DES" Advances in Cryptology -- CRYPTO '96 237-251.

[JohnBruceDavid98] J Kelsey, B Schneier, D Wanger "Related Key Cryptanalysis of 3-way, Biham DES, CAST, DES-X, New DES, RC2 and TEA"

[Kahn67] D. Kahn, The Codebreakers: The Story of Secret Writing, New York: Macmillan Publishing Co., 1967.

[KamDavid1979] Kam, J. and G. Davida. 1979. Structured Design of Substitution-Permutation Encryption Networks. IEEE Transactions on Computers. C-28(10): 747-753.

[Lloyd90] Lloyd, S. 1990. Properties of binary functions. Advances in Cryptology -- EUROCRYPT '90. 124-139.

[Matsui93] Matsui, M. 1993. Linear Cryptanalysis Method for DES Cipher. Advances in Cryptology -- EUROCRYPT '93. 386-397.

[Matsui94] Matsui, M. 1994. The First Experimental Cryptanalysis of the Data Encryption Standard. Advances in Cryptology -- CRYPTO '94. 1-11.

[MatsuiYamagishi94] Matsui, M. and A. Yamagishi. 1994. A New Cryptanalytic Method for FEAL Cipher. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Science. E77-A(1): 2-7.

[MeierStaffelbach89] Meier, W. and O. Staffelbach. 1989. Nonlinearity Criteria

for Cryptographic Functions. Advances in Cryptology -- EUROCRYPT '89. 549-562.

[Nyberg Knudson92] Nyberg, K. and L. Knudsen. 1993. Provable Security Against Differential Cryptanalysis. Advances in Cryptology -- CRYPTO '92. 566-574.

[Nyberg91] Nyberg, K. 1991. Perfect nonlinear S-boxes. Advances in Cryptology -- EUROCRYPT '91. 378-386.

[Pieprzyk89A] Pieprzyk, J. 1989. Error propagation property and application in cryptography. IEE Proceedings, Part E. 136(4): 262-270.

[Pieprzyk989B] Pieprzyk, J. 1989. Non-linearity of Exponent Permutations. Advances in Cryptology -- EUROCRYPT '89. 80-92.

[PieprzykFinkelstein88] Pieprzyk, J. and G. Finkelstein. 1988. Towards effective nonlinear cryptosystem design. IEE Proceedings, Part E. 135(6): 325-335

[PieprzykFinkelstein89] Pieprzyk, J. and G. Finkelstein. 1989. Permutations that Maximize Non-Linearity and Their Cryptographic Significance. Computer Security in the Age of Information. 63-74.

[PLLGVandewalle90] Preneel, B., W. Van Leekwijck, L. Van Linden, R. Govaerts and J. Vandewalle. 1990. Propagation Characteristics of Boolean Functions. Advances in Cryptology -- EUROCRYPT '90. 161-173.

[Rainier86] Rainier A. Rueppel (1986), "Analysis and Design of Stream Ciphers," Spreinger Verlag.

[SivaTavPeppard92] Sivabalan, M, S. Tavares and L. Peppard. 1992. On the Design of SP Networks from an Information Theoretic Point of View. Advances in Cryptology -- CRYPTO '92. 260-279.

[STACA] William Stalling Fifth Edition Computer Organization and Architecture , Prentice Hall Inc

[Vaudenay95] Vaudenay, S. 1995. An Experiment on DES Statistical Cryptanalysis

[WebsterTavares85] Webster, A. and S. Tavares. 1985. On the Design of S-Boxes. Advances in Cryptology -- CRYPTO '85 523-534.

[WheelerNeedham94] In Bart Preneel, editor, Fast Software Encryption: Second International Workshop, volume 1008 of Lecture Notes in Computer Science, pages 363-366, Leuven, Belgium, 14-16 December 1994. Springer-Verlag.

[YoussefTavares95A] Youssef, A., S. Tavares. 1995. Number of Nonlinear Regular S-boxes. IEE Electronics Letters. 31(19): 1643-1644.

[YoussefTavares95B] Youssef, A., S. Tavares. 1995. Resistance of Balanced S-boxes to Linear and Differential Cryptanalysis. Information Processing Letters. 56: 249-252.

[YoussefTavares95C] Youssef, A., S. Tavares. 1995. Information Leakage of a Randomly Selected Boolean Function. Proceedings of the 4th Canadian Workshop on Information Theory. Lecture Notes in Computer Sciences (LNCS) 1133. Springer-Verlag. 41-52.

[YouTavMisAdam95] Youssef, A., S. Tavares, S. Mister and C. Adams. 1995. Linear Approximation of Injective S-boxes. IEE Electronics Letters. 31(25): 2168-2169

[ZhangZheng95] Zhang, X. and Y. Zheng. 1995. GAC -- the Criterion for Global Avalanche Characteristics of Cryptographic Functions. Journal for Universal Computer Science. 1(5): 316-333.

From : ANSInet Publications <sarwarm@cityonline.net.pk>

Sent : Monday, July 19, 2004 8:18 AM

To : <amerhayat2@hotmail.com>

Subject : Acknoledgement: 169-PJIT

INFORMATION TECHNOLOGY JOURNAL
A Quarterly Publication of ANSInet

Khawaja Amer Hayat
Department of Computer Science,
International Islamic University Islamabad,
Pakistan

Subject: Acknoledgement 169-PJIT

Dear Khawaja Amer Hayat,

Thank you for your interest in ANSInet Publications

We have received a manuscript entitled "Password interception in a SSL/TLS Channel" for publication in INFORMATION TECHNOLOGY JOURNAL.

Serial # of the manuscript is 169-PJIT.

Please remember and mention this manuscript # in all future correspondence.

I wish you a very nice day.

Regards

Muhammad Imran Pasha
Account Manager

Asian Network for Scientific Information, 433, St. 5, Sarfraz Colony, Fowara Chowk, Faisalabad-Pakistan
Tel: +92(0)303 6703633; Fax: +92 (0)41 731433 E-mail: accountmanager@ansinet.org

From : ANSInet Publications <sarwarm@cityonline.net.pk> | | | Inbox
Sent : Wednesday, July 21, 2004 9:53 AM
To : <amerhayat2@hotmail.com>
Subject : Comments on Article: 169-PJIT

INFORMATION TECHNOLOGY JOURNAL
A Quarterly Publication of ANSInet

Khawaja Amer Hayat
Department of Computer Science,
International Islamic University Islamabad,
Pakistan

Subject: Comments on Article # 169-PJIT

Dear Khawaja Amer Hayat,

First of all I would like to thank you for your trust on INFORMATION TECHNOLOGY JOURNAL.

We have received the referees comments and according to them your manuscript has been accepted for publication in INFORMATION TECHNOLOGY JOURNAL after suggested modification.

Please download the referee`s evaluated copy from the following link.

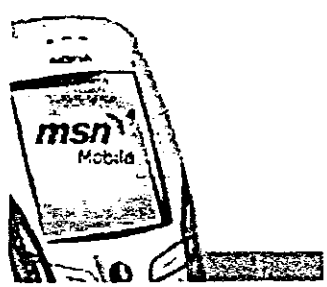
http://64.4.34.250/cgi-bin/linkrd?_lang=EN&lah=bb00e1ceecc06d266bedc14a5b4d71a1&lat=1090692725&hm___action=http%3a%2f%2fwww%2eansinet%2enet%2fpapers%2f169%2dPJIT%2ezip

How to download the file
1) Press right button on the above mentioned link
2) Select option "Save Target as ...."
3) Choose path where the file should be saved.

It is requested to inform us after downloading the file.

Your quick response will be highly appreciated.

With best regards



Muhammad Imran Pasha
Account Manager

Asian Network for Scientific Information, 433, St. 5, Sarfraz Colony, Fowara Chowk,
Faisalabad-Pakistan
Tel: +92(0)303 6703633; Fax: +92 (0)41 731433 E-mail: accountmanager@ansinet.org

Password interception in a SSL/TLS Channel

[1]Khawaja Amer Hayat , [2]Umar Waqar Anis , [3]Tauseef Ur Rahman,
[1,2]Department of Computer Science, International Islamic University Islamabad. Pakistan
[3]Department of Telecom and Computer Engineering, International Islamic University Islamabad, Pakistan
{[1]amer@iiu.edu.pk,[2]umar@iiu.edu.pk,[3]tauseef@iiu.edu.pk }

Abstract:. This study represents the attack and demonstrates the optimized form of it. It works against the latest and most popular implementations of SSL/TLS for password interception. We show that a password for an IMAP account can be intercepted when the eavesdropper is near the server in an hour of processing. It can be concluded that the versions of SSL/TLS are insecure when used with block ciphers in CBC mode. Before conclusion we will show the conditions for the attack. In the end this study propose ways to make these versions strong and much more secure.

Key words: IMAP account, RFC 2246, MAC.

Introduction

Cipher-Block chaining mode

CBC mode processes the data based on some initialization vector IV which is a l-bit string. The input to the encryption algorithm is the XOR of the current plaintext block and the preceding cipher text block.

Secure Socket Layer (SSL) and Transport Layer Security (TLS)

SSL was originated by Netscape. TLS working group was formed within IETF. First version of TLS can be viewed as an SSLv3.1. Fig 1 shows the record format of SSL.

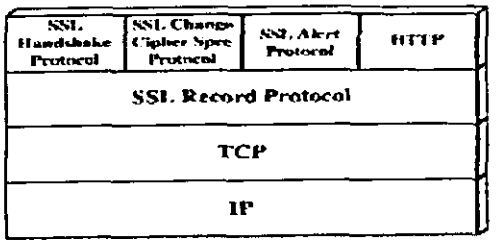


Fig 1: SSl Protocol Stack

Figure 2 shows the operation of the SSL Record Protocol.

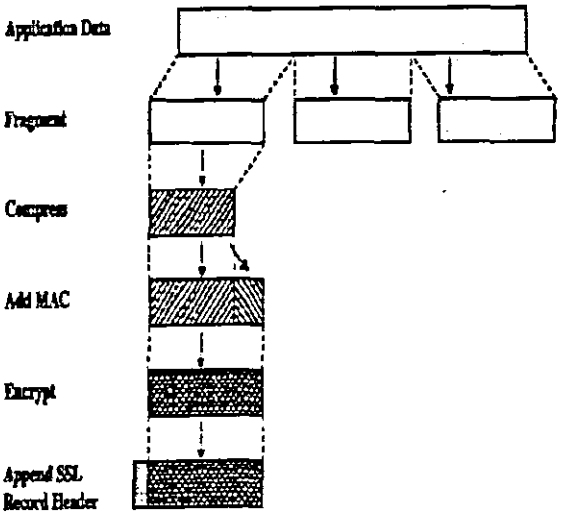


Fig 2: SSL Record Protocol

TLS has the same record format as the SSL . it is defined in RFC 2246 and is Similar to SSLv3. the differences between SSL and TLS are

1. version number
2. message authentication code
3. pseudorandom function
4. alert codes
5. cipher suites
6. client certificate types
7. certificate verify and finished message
8. cryptographic computations
9. padding

CBC-PAD in Secured Channels

Peer-to-peer secure and safe channels can be established by the Transport Layer Security Protocol. It requires,

- Negotiating a cipher suite and security parameters between the two engaged parties.
- Exchanging secret keys between the parties engaged in communication.
- Then messages M are first authenticated with a Message Authentication Code (MAC).
- Then encrypted with a symmetric cipher. Block ciphers, e.g. the Triple DES (3DES) are often used in Cipher Block Chaining (CBC) mode with padding.

Let b be the block length in characters let b = 8 for DES as it requires 64-bit M. Let M be the message to be sent. First we append the MAC of M to M. We obtain MjMAC. Then we pad MjMAC with a padding PAD such that MjMACjPADjLEN is of length a multiple of b where LEN is a single byte whose value ' is the length of PAD in bytes. PAD is required by the TLS specifications to consist of l bytes equal to l. Then MjMACjPADjLEN is cut into a block sequence x 1, x 2 ..., x n (each x i has a length of b), then encrypted in CBC mode, i.e. changed into y 1 ,,y 2 ,.... y n with

$$M_i=ENC(y_{i-1} \text{ xor } x_i)$$

where ENC denotes the block cipher. The initial vector IV y_0 can be either a part of the secret key, or a random value sent with the cipher text, or a xored value. When y 1 , y 2 , ..., y n is received, it is first decrypted back into x 1 , x 2 ...., xn . Then we look at the last byte LEN, call l its value, and separate the padding PAD of length l and LEN from the plaintext. It is required that PAD should be checked to consist of bytes all equal to l. If this is not the case, a padding error is generated. Otherwise the MAC is extracted then checked. If the MAC is not valid, a MAC error is generated. Otherwise the clear text M is extracted and processed. In TLS, fatal errors such as incorrect padding or bad MAC errors simply abort the session. The error messages are sent through the same channel, i.e. they are MACed, padded, then encrypted before being sent. A typical application of TLS is when an email application connects to a remote IMAP server. For this, the client simply sends the user name and password through the secured channel, i.e. the message M includes the password in clear.

Side Channel Attack against CBC-PAD

It assumes that we can send a cipher text to the server and get the answer which is either an error or an acknowledgment. We model it as an prediction P. When the answer is a padding error message (decryption failed), we say that P answers 0. Otherwise the prediction P returns 1. Let y be the cipher text block to decrypt. The purpose of the attack is to find the block x such that y = ENC(x). We first transform the prediction P into an prediction Check1(y; u) which checks whether the ENC 1 (y) block ends with the byte sequence u" or not. We then use this prediction in DecryptByte1(y; s) in order to decrypt a new character in ENC 1 (y) from the known tail s of x. We then use this process in DecryptBlock1(y) in order to decrypt a full block y. The attack of [1] works against WTLS [2]. It does not work against TLS for two reasons. First of all, as soon as a padding or MAC error occurs the session is broken and needs to restart with a freshly exchanged key. As pointed out in [1], the attack could have still worked in order to decrypt only the rightmost byte with a probability of success of 2^{-8}. It can also be adapted in order to test if x ends with a given pattern. This does not work either against TLS for another reason i.e. error messages are not available to the antagonist (they are indeed encrypted and indistinguishable). In order to make them even less distinguishable", standard implementations of the TLS protocol now use the same error message for both types or errors (as specified for SSL) in order to protect against this type of attack [3].

DecryptBlock1(y)
1: for i = 1 to b do
2: c_i = DecryptByte1(y; c_{i+1} | ...| c_l)
3: end for
4: return c_b | ...| c_l

DecryptByte1(y, s)
1: for all possible values of byte c do
2: if Check1(y, c | s) = 1 then
3: return c
4: end if
5: end for

Check1(y, u)
1: let i be the length of u
2: let L be a random string of length b i
3: let R = (i -1)| (i -1)|...|(i -1) of length i
4: r = L|(R⊕ u)
5: build the fake cipher text r | y to be sent to the prediction

6: return P(r | y)

Fig 3 : Side Channel Attack against CBC-PAD.

Timing Attack

Attack Principles

In order to get access to the error type which is not directly available, we try to deduce it from a side channel by performing a timing attack [6]. Instead of getting 0 or 1 depending on the error type, we now have prediction which outputs the timing answer T of the server. The principle of the attack is as follows: in order to check if the padding is correct, the server only needs to perform simple operations on the very end of the cipher text. When the padding is correct, the server further needs to perform cryptographic operations throughout the whole cipher text in order to check the MAC, and this may take more time. We use the variation between the time it takes to perform the two types of operations in order to get the answer from the prediction P.

We increase the discrepancy of the two types of errors by enlarging the cipher text: the longer the cipher text, the longer the MAC verification. (The MAC verification time increases linearly with the length of MES.) Hence we replace the r | y fake cipher text in DecryptByte1 by f |r| y where f is a random block sequence of the longest acceptable length (i.e. 2 14 + 2048 bytes in TLS).

On Fig. 4 is the updated algorithm. It uses a DecryptBlock2 algorithm which is similar to fig 3. Note that Check2 may miss the right byte, so DecryptByte2 needs to repeat the loop until the byte is found.

Experiment

We made a statistical analysis of the answer time for the two types of errors. The expected values μ_R and μ_w and the standard deviations μ_R and μ_w for the two distributions are as follows:

$\mu_R \sim 23{:}63 \quad \mu_w \sim 21{:}57$

$\sigma_R \sim 1{:}48 \quad \sigma_w \sim 1{:}86$

Note that these values were obtained on a LAN where a fire wall was present between the attacker and the server, so the attacker was not directly connected to the server.

DecryptByte2(y| s)

1: repeat
2: for all possible values of byte c do
3: if Check2(y, c | s) = 1 then
4: return c
5: end if
6: end for
7: until byte is found

Check2(y | u)

1: make r in order to test u as in Check1
2: build the fake cipher text f |r| y to be sent to the prediction
(f is the longest possible random block sequence)
3: query the prediction n times and get T1 ..., T n
(answers which are larger than B are ignored)
4: return ACCEPT(T1 ,..., T n)

Fig. 4. Regular Timing Attack.

Analysis of the Best ACCEPT Predicate

The ACCEPT predicate is used in order to decide whether the distribution of the answers is DR (the predicate should be true) or DW (the predicate should be false). The predicate introduces two types of wrong information. We let $\epsilon+$ (resp. $\epsilon-$) be the probability of bad decision when the distribution is D_W (resp. D_R). The ϵ + and ϵ- probabilities can be interpreted as the probabilities of false positives and false negatives of a character correctness test. The optimal tradeoff between ϵ + and ϵ- is achieved by the ACCEPT predicate which is given by the Neyman-Pearson lemma:

$$\text{ACCEPT} : \frac{f_R(T_1)}{f_W(T_1)} \times \ldots \times \frac{f_R(T_n)}{f_W(T_n)} > ?$$

with f_R and f_W the density functions of DR and DW respectively and a given threshold r .
Depending on r we trade ϵ + against ϵ . With the approximation by a normal distribution the ACCEPT test can be written

$$\frac{T_1+\ldots+T_n}{n} > \frac{\mu_R+\mu_W}{2} + \frac{\sigma^2 \log \tau}{n(\mu_R-\mu_W)}$$

so we equivalently can change the definition of τ and consider

$$\text{ACCEPT}: \frac{T_1+\ldots+T_n}{n} > \tau'$$

as the ACCEPT test choice. With this we obtain

$$\varepsilon_- = \varphi\left(\frac{\tau'-\mu_R}{\sigma}\sqrt{n}\right)$$

$$\varepsilon_+ = \varphi\left(-\frac{\tau'-\mu_W}{\sigma}\sqrt{n}\right)$$

where

$$\varphi(z) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{z} e^{-\frac{t^2}{2}}dt.$$

Using Sequential Decision Rules

The following algorithm shows a more general skeleton. Basically, we collect timing samples T j until some STOP predicate decides that there are enough of these for the ACCEPT predicate to decide. We use DecryptByte3 fig 5 and DecryptBlock3 algorithms which are similar to DecryptByte2 and ecryptBlock2.

Check3(y, u)
1: make r in order to test u as in Check1
2: build the fake cipher text f |r| y to be sent to the prediction as in Ckeck2
3: j = 0
4: repeat
5: j ++
6: query the prediction and get T j
(a T j larger than B is ignored and the query is repeated)
7 until STOP(T1 ,..., T j)
8: return ACCEPT(T1 ,..., T j)

Fig 5: Timing Attack with a Sequential Distinguisher.

Multi-session Attack

Attack Strategy

Since sessions are broken as soon as there is an error, the attacks from previous sections do not work. We now assume that each TLS session includes a critical plaintext block x which is always the same (e.g. a password) and that we intercept the corresponding cipher text block y = ENC(x y '). (Here y 0 is the previous cipher text block following the CBC mode.) The target x, is constant in every session, but y and y 0 depend on the session. The full attack is depicted in the algorithm discussed in analysis. Here the Check4 prediction no longer relies on some y since this block is changed in every session. The Check4(u) is called in order to check whether \the x plaintext block ends with the byte sequence u" or not. The plaintext block x is equal to ENC $^{-1}$ (y) y ' for some current key, and some current cipher text blocks y and y' . We assume that the prediction can get y and y ' .

Analysis

Let C be the average complexity of DecryptBlock4. Let Z denote the set of all possible byte values. Let p be the probability of success of DecryptBlock4.

Let p i be the success probability of DecryptByte4(s) assuming that s is the right tail of length i-1. We have p = p 1 ... p b .

In order to simplify our analysis, we assume that the target block is uniformly distributed in Z b so that step 1 of DecryptByte4 can be ignored. We further consider a weaker algorithm in which the outer repeat/until loop of DecryptByte4 is removed (i.e. we consider that the attack fails as soon as a STOP predicate is satisfied but the ACCEPT predicate takes a bad decision).

DecryptBlock4
1: for i = 1 to b do
2: c i DecryptByte4(c i -1 |...| c 1)
3: end for
4: return c b |...| c 1

DecryptByte4(s)
1: sort all possible c characters in order of decreasing likelihood.
2: repeat
3: for all possible values of character c do
4: if Check4(c| s) = 1 then
5: return c
6: end if
7: end for
8: until byte is found

Check4(u)
1: j= 0
2: repeat
3: j ++
4: wait for a new session and get the current y and y 0 blocks

5: let i be the length of u
6: let L be a random string of length b i
7: let R = (i -1)j(i -1)|...| (i- 1) of length i
8: r = (L| (R⊕u)⊕ y '
9: build the fake cipher text f |r| y to be sent to the prediction
(f is the longest possible random block sequence)
10: query the prediction and get T j
(if it is larger than B then go back to Step 4)
11: until STOP(T1 ,...,T j)
12: return ACCEPT(T1,..., T j)
Fig 6:Password Interception inside SSL/TLS.

Password Interception with Dictionary Attack

Attack Description

We now use the a priori distribution of x in the previous attack in order to decrease the complexity. For instance, if x is a password corresponding to an IMAP authentication. we perform a kind of dictionary attack on x. We assume that we have pre computed a dictionary of all possible x blocks with the corresponding probability of occurrence. We use it in the first step of DecryptByte4 in order to sort the c candidates.

Analysis

We consider a list of possible blocks $c_b \dots c_1$. We let $Pr[c_b \dots c_1]$ be the occurrence probability of a plaintext block. We also let $Pr[c_i \dots c_1]$ be the sum $Pr[c_b \dots c_1]$ for all possible $c_b \dots c_{i+1}$. We arrange the dictionary of all blocks into a search tree. The root is connected to many sub trees. each corresponding to a c_1 character. Each sub tree corresponding to a c_1 character is connected to many sub-sub trees, each corresponding to a c_2 character... We label each node of the tree by a c i ... c 1 string. We assume that the list of sub trees of any node c i ... c 1 is sorted in decreasing order of values of Pr[c i+1 c i ... c 1]. We let $N(c_{i+1} \dots c_1)$ be the rank of the $c_{i+1} \dots c_1$ sub tree of the node c i ... c_1 in the list.

Numerical Example

We have used dictionary [5] from which we have selected only words of size b = 8 characters (i.e. 8 bytes), giving a total word count of 712 ' 786 words and ordered it as described the previous section. For this dictionary, we have calculated that C 0 = 31 and then implemented algorithm DecryptBlock4 and confirmed this result. Note that $C_0 = 31$ is a quite remarkable result since the best search rule for finding a password out of a dictionary of D = 712 0 786 words consists of $\lfloor \log_2 D \rfloor = 20$ binary questions, so the overhead is only of 11 questions.

Implementation of the Attack

Here we describe how the DecryptBlock4 was implemented in practice against an IMAP email server.

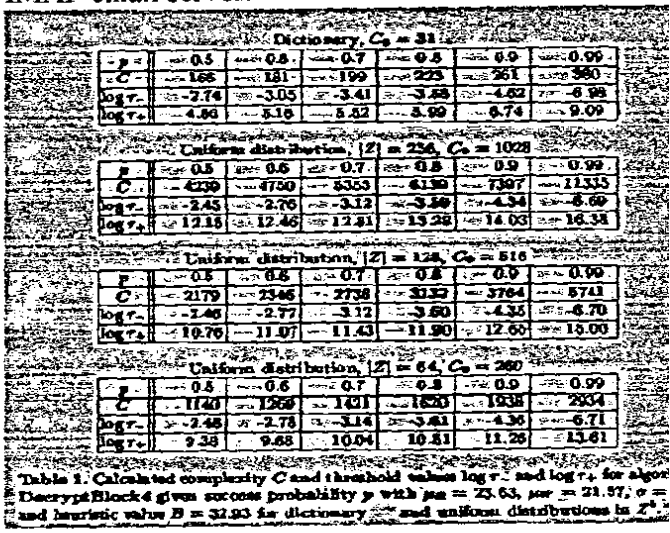


Table 1. Calculated complexity C and threshold values $\log \tau_-$ and $\log \tau_+$ for algorithm DecryptBlock4 given success probability p with $\mu_R = 23.63$, $\mu_W = 21.57$, $\sigma = 1.5$ and heuristic value $B = 32.93$ for dictionary Z^* and uniform distributions in Z^*.

Setup

The multi-session attack has been implemented using the Outlook Express 6.x client from Microsoft under Windows XP and an IMAP Rev 4 server 8 . Outlook sends the login and password to the IMAP server using the following format:

XXXX LOGIN "username" "password"<0x0d><0x0a>

Here XXXX are four random digits which are incremented each time Outlook connects to the server. An interesting feature of Outlook is that (by default) it checks for messages automatically every 5 min and also that it requires an authentication for each folder created on the IMAP user account. i.e. we have a bunch of free sessions every 5 min. For instance. with five folders (in, out, trash, read, and draft), we obtain 60 sessions every hour. If Outlook is now configured to check emails every min. the fastest attack of Table 1 with 166 sessions requires half an hour. Outlook notices that some protocol errors occur but this does not seem to bother it at all. The TLS tunneling between the IMAP server and Outlook Express was implemented using stunnel v3.22 9 .

This is a man-in-the-middle type attack where connection requests to the IMAP server from the Outlook client are redirected to the attacker's machine using DNS spoofing where the attacker intercepts the authentication messages and attempts to decrypt it using DecryptBlock4. Note that the attack is performed on a Local Area Network.

Conditions for the attack

Obviously, the attack works if the following conditions are met.

- A critical piece of information is repeatedly encrypted at a predictable place.
- A block cipher in CBC mode is chosen.
- The attacker can sit in the middle and perform active attacks.
- The attacker can distinguish time differences between two types of errors.

Conclusion

We have derived a multi-session variant of the attack [1] in order to show that it is possible to attack SSL/TLS in the case when the message that is being encrypted remains the same during each session. This is the case, for example, when an email client such as Outlook Express connects to an IMAP server. We have detailed the attack and described the setup we have used in order to perform it. One problem we have encountered is that the error messages sent in SSL/TLS are encrypted and it is not possible to easily differentiate which is being sent by the client or the server. A solution to this problem is to look at timings between errors messages.

References

[1] Vaudenay , 2002 Security Flaws Induced by CBC Padding | Applications to SSL, IPSEC, WTLS... In Advances in Cryptology EUROCRYPT'02, Amsterdam, Netherland, Lectures Notes in Computer Science 2332, pp. 534-545.

[2] Wireless Transport Layer Security. Wireless Application Protocol 2001,WAP-261- WTLS-20010406-a. Wireless Application Protocol Forum, http://www.wapforum.org/

[3] FIPS 46-3, 1999 Data Encryption Standard (DES). U.S. Department of Commerce | National Institute of Standards and Technology. Federal Information Processing Standard Publication 46-3,.

[4] FIPS 81, 1980,DES Modes of Operation. U.S. Department of Commerce | National Bureau of Standards, National Technical Information Service, Springeld, Virginia. Federal Information Processing Standards 81,.

[5] English Word List Elcomsoft Co. Ltd. http://www.elcomsoft.com

[6]P.Kocher.Timing Attacks on Implementations of Di/Ee-Hellman, RSA, DSS, and other Systems. 1996. In Advances in Cryptology CRYPTO'96, Santa Barbara, California, U.S.A., Lectures Notes in Computer Science 1109, pp. 104-113.

From : ANSInet Publications <sarwarm@cityonline.net.pk> | Inbox
Sent : Wednesday, July 21, 2004 9:53 AM
To : <amerhayat2@hotmail.com>
Subject : Comments on Article: 170-PJIT

INFORMATION TECHNOLOGY JOURNAL
A Quarterly Publication of ANSInet

Khawaja Amer Hayat
Department of Computer Science,
International Islamic University Islamabad,
Pakistan

Subject: Comments on Article # 170-PJIT

Dear Khawaja Amer Hayat,

First of all I would like to thank you for your trust on INFORMATION TECHNOLOGY JOURNAL.

We have received the referees comments and according to them your manuscript has been accepted for publication in INFORMATION TECHNOLOGY JOURNAL after suggested modification.

Please download the referee`s evaluated copy from the following link.

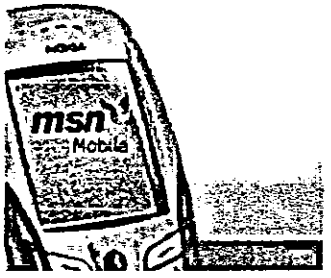
http://64.4.34.250/cgi-bin/linkrd?_lang=EN&lah=dabe8cee612afad0c46d72e0a9ad9508&lat=1090692659&hm___action=http%3a%2f%2fwww%2eansinet%2enet%2fpapers%2f170%2dPJIT%2ezip

How to download the file
1) Press right button on the above mentioned link
2) Select option "Save Target as ...."
3) Choose path where the file should be saved.

It is requested to inform us after downloading the file.

Your quick response will be highly appreciated.

With best regards



Muhammad Imran Pasha
Account Manager

Asian Network for Scientific Information, 433, St. 5, Sarfraz Colony, Fowara Chowk, Faisalabad-Pakistan
Tel: +92(0)303 6703633; Fax: +92 (0)41 731433 E-mail: accountmanager@ansinet.org

Suppose we have some unknown number z. we a re allowed to add any number we like modulo 2^{64} and then xor it with another number of our choosing we choose T and U and test whether $(z+T \bmod 2^{64})$ xor $U = Z$

We can learn the value of z with enough queries. we can add to k1, and xor u into our plain text block or voce versa. One form of attack uses T and U values with the same single bit on. If it results in the same cipher text when resulted t=U=0, then we learn that bit in K1 was a one. There is another suggestion [15] using a DES-x variant which replaces the xor pre- and post whitening steps by addition modulo 2^{64}:

C=k1+DESk2 (K3+P). It shows that it is vulnerable to a related key attack very similar to the one that works against regular DES-X.

Biham-DES

Biham and Biryukov proposed that by modifying the F function slightly by using extra bits DES can be made much stronger [16]. One of these propositions uses 5 key bits to select from 32 possible reordering of the 8 DES S-boxes. Suppose one key uses ordering 15642738 and another uses ordering 75642138.the only difference between the two F-functions is that S-boxes 1 and 7 have been swapped.

Observe that

$$Pr_x(S1[s] \text{ xor } S7[x \text{ xor } 2]=0) = 14/64.$$

the input differential 2 appears only in the middle input bits of the S-box, and will not spread to neighboring S boxes. it shows that construction of a one round property with probability $(14/64)^2$. it leads to a 13 round iterative property with chance $(14/64)^{12} = 2^{-26}$.the differential techniques of Biham and Shamir [17] will break Biham DES with 2^{27} chosen plaintexts when this special[1] related-key pair is available. One useful pair of related key partners can be obtained from any starting key after 32 related key queries. when using this with a 32 recommended DES S-box reordering, we have a 1/16 probability of success when 2^{27} chosen plaintexts and one related key query are available; success is nearly guaranteed with 2^{31} chosen plain texts and 32 related -key queries.

Tiny Encryption Algorithm

TEA [18] is a festal block cipher and uses a 128 bit master key's [0...3].also it requires a simple key schedule. Odd rounds use k [,01] and even rounds use k[2,3] as a round sub key. Two rounds of TEA applied to the block Ai, Bi consists of c=c+&

$$Ai+1 = Ai + F(Bi, k[,1],c) \quad Bi+1 = Bi + F(Ai+1,k[j]).$$

here SL4(z) denotes the result of shifting z to the left 4 bits and Sr is shift right .c is a value which perturbs the F function so that it is different in each round to avoid degenerated attacks. C is initially 0 and is incremented by & = [(underroot5-1)2^{31}].Due to simplicity in key schedule. it is vulnerable it related key attacks.

Attack One

Using differential RK attack, consider the affect of simultaneously flipping bit 30 of K[2] and K[3].With probability 0.5, output of the F function in the even rounds will remain the same. Our analysis indicates that a 4R differential RK attack a break 64- round TEA with one RK query and about 2^{34} chosen plain texts. This is only one of several of this type of property.

Attack Two

Other form of RK attack is very similar to the first. we request the encryption of (Y,Z) under key K[0..3] and the encryption of (Y,Z xor 2^{31}) under key k*[0..3] xor (0,2p31 xor 2^{26},0,0).If we observe thee terms of F (Z,K[0,1],c) when bit 31 of Z is flipped along with bits 26 and 31 of K[1], we find that

SL4 (Z) +K [0] Z+c Sr5 (Z) +K[1]

This gives a one cycle (2round) iterative differential property with chance of success 0.5, when we can choose one key difference.

Suggestions for Key-schedule Design

There are some similarities between the requirements for a strong key schedules and cryptographic hash functions. Some rules are

- Key schedule should be hard to invert or recover any information about other bits in the key. Hash functions are also one way.
- Collision freedom is a requirement in key schedule and a property of a hash function.
- It should not be possible to produce controlled changes in round keys.

Key schedules of blowfish [19] and SEAL [20] used the same principle. Incorporating linear key schedules DES appears to be a subtle and difficult goal to achieve. In several ciphers linear key schedules have been proved to be weak. Protection against sub key rotation attacks [2]

Workshop Proceedings, Springer-Verlag. pp. 56-63.
[21] R.Winternitz and M.Hellman, 1994, "Chosen-key Attacks on a Block Cipher," Cryptologia v.11.n,1 pp16-20

Cryptanalysis of some encryption/ciphers schemes using related key Attack.

[1]Khawaja Amer Hayat , [2]Umar Waqar Anis , [3]Tauseef Ur Rahman,
[1,2]Department of Computer Science, International Islamic University Islamabad, Pakistan
[3]Department of Telecom and Computer Engineering, International Islamic University Islamabad, Pakistan
{[1]amer@iiu.edu.pk,[2]umar@iiu.edu.pk,[3]tauseef@iiu.edu.pk }

Abstract: This paper represents some Related key attacks on the block ciphers like Biham-DES, DES-X, New DES and 3 way attacks and also try to deal individual by adopting general attacking scheme. The study includes some suggestions to protect one from such attacks.

Key Words: Related key attack , Hash Functions, Exhaustive attacks, Festal block Ciphers, collision Freedom, Differential RK attacks.

Introduction

Related key attack works on the assumption that *third party learns cipher text of certain plaintext* under the original key and also under the derived key k'. Such that k'= f (k). Other form of attack called chosen related key attack; the attacker specifies how the key is to transform. Difference between chosen related key attack and known related key is that in known RK attack the key difference is known but not at the will of attacker. He chooses between the keys rather than the actual ones. These techniques have been developed in [1-3].the attacker may change few bits in the key without knowing the key and also the key update protocols using known functions like k,k+2,k+4 etc. Rotor machines were also hit by the same attack[4].Hash functions built from block ciphers are also vulnerable through this attack.[5-6].There are practical protocols that allow related key attacks to be mounted[3] against block cipher.[3] presented related key attacks against GOST.[7], IDEA [8], SAFER K-64[9], G-DES [10-11] and triple DES.

New Differentials Related Key Attacks

3-WAY

It uses 11 rounds cipher scheme and requires 96-bit blocks [12].it has a round function f(x) which is equitant to y=N(x), z= L(y), F(x) = z xor k xor c where N is a static nonlinear layer built out of 32 parallel 3-bit permutation S-boxes is a linear function which is fixed, k is a 96 bit key and c is also static.

3-WAY *is vulnerable to a related key differential* attack. it is easy to figure out a differential characteristic Δx-Δy with probability 0.25 so we can construct a characteristic Δx to Δy with probability 0.25 for nonlinear layer N by using only 1 active S-BOX Linearity show s that Δy to Δz= L (Δy) with probability 1 under the linear layer L. by picking $\Delta k=\Delta x$ xor Δz we have Δx to Δx by F with probability 0.25 which is one round *iterative differential property.* by adopting a 9 round property with probability 2-18 to cover these rounds and applying 2R analysis to the last 2 rounds 3-WAY can be broken by one related key query and approximately 2^{22} chosen plain texts.

DES- X

Rivest[13] proposed a variant of DES called DES-X which is stronger against exhaustive attacks. Using three keys (k1, K2, k3) it converts P to C by the following way.

C=k1 xor DES K2 (k3 xor P) where k3 is pre whitening key and k1 is a post whitening key. This scheme has many complementation properties. every key (k1, k2, k3) has its corresponding complementary keys (k1', k2', k3').This characteristic leads to an attack which requires $2^{56+64-n}$ trial encryptions when 2^n chosen plain texts are available [14]. We give a related key differential; attack on DES-X using key differences modulo 2^{64} and plain text difference modulo 2. The attack requires 64 chosen key relations to recover the key, with one plain text encrypted under each new key.

can be achieved by generating sub keys differently. As a result each key bit affects nearly every round, but not in the same way. Also key schedules should be specially designed to avoid *differential RK attacks. If related* queries are cheap, the master key should be long to avoid generic black box attacks.[21],[3]. Make sure that each bit has equal power and effect on the round keys. Independent round sub keys should be avoided as it increases cipher's key length and decreases its resistance against such attacks. At the end protocol designers should be aware of RK attacks and they should design tamper-resistant hardware to avoid changes in the sub key without such changes being detected.

Results

Our attacks are build on the original study showing how to adopt the general attack to deal with the difficulties of the individual algorithms namely Biham-DES, DES-X, New DES and 3 *way . Our attacks prove that the security of these* algorithms can be compromised by exploiting the weaknesses in these algorithms. This study also suggested some design principles to protect against these attacks.

References

[1] L.R. Knudsen, 1993, "Cryptanalysis of LOKI91," Advances in Cryptology--AUSCRYPT '92, Springer-Verlag, pp. 196-208.

[2] E. Biham, 1994,"New Types of Cryptanalytic Attacks Using Related Keys," Advances in Cryptology—EUROCRYPT'93,Springer-Verlag, pp. 398-409.

[3] J. Kelsey, B. Schneier, and D. Wagner, 1996,"Key-Schedule Cryptanalysis of IDEA, G-DES, GOST, SAFER, and Triple-DES," Advances in Cryptology--CRYPTO '96, Springer-Verlag, pp. 237-251.

[4] W. Diffie and M.E. Hellman. March 1979 "Privacy and Authentication: An Introduction to Cryptography". Proceedings of the IEEE, vol 67 no 3,.

[5] R. Winternitz, 1984,"Producing One-Way Hash Functions from DES," Advances in Cryptology: Proceedings of Crypto 83, Plenum Press, pp. 203-207.

[6] Research and Development in Advanced Communication Technologies in Europe, Jun 1992 RIPE *Integrity Primitives: Final Report of* RACE Integrity Primitives Evaluation (R1040), RACE,.

[7] GOST, Gosudarstvennyi Standard 28147-89, 1989,"Cryptographic Protection for Data Processing Systems." Government Committee of the USSR for Standards.

[8] X. Lai, J. Massey, and S. Murphy, 1991,"Markov Ciphers and Differential Cryptanalysis," Advances in Cryptology--CRYPTO '91, Springer-Verlag, pp. 17-38.

[9] J.L. Massey, 1994,"SAFER K-64: A Byte-Oriented Block-Ciphering Algorithm", Fast Software Encryption, Cambridge Security Workshop Proceedings, Springer-Verlag, pp. 1-17.

[10]A.Pfitzmann and R. Abmann,1990,"Efficient Software Implementations of (Generalized) DES," Proc. SECURICOM '90, Paris, pp. 139-158.

[11] A. Pfitzmann and R. Abmann, 1990."More Efficient Software Implementations of (Generalized) DES," Technical Report PfAb90, Interner Bericht 18/90, Fakultat fur Informatik, UniversitatKarlsruhe,http://www.informatik.unih ildesheim.de/~sirene/lit/abstr90.html#PfAss 90

[12] J. Daemen, 1994, "A New Approach to Block Cipher Design," Fast Software Encryption, Cambridge Security Workshop Proceedings, Springer-Verlag, pp.18

[13] R. Rivest, personal communication.

[14] J. Daemen, 1992, "Limitations of the Even-Mansour Construction," Advances in Cryptology--ASIACRYPT '91, Springer-Verlag. pp. 495-498.

[15] J. Kilian and P. Rogaway, 1996, "How to protect DES against exhaustive key search," Advances in Cryptology--CRYPTO '96. Springer-Verlag, pp. 252-267

[16] E. Biham and A. Biryukov,1994, "How to Strengthen DES Using Existing Hardware," Advances in Cryptology--ASIACRYPT '94, Springer-Verlag, pp. 398-412.

[17] E. Biham and A. Shamir,1993 "Differential Cryptanalysis of the Full 16-round DES." Advances in Cryptology--CRYPTO '92. Springer-Verlag pp. 487-496.

[18] D. Wheeler and R. Needham, 1995,"TEA, a Tiny Encryption Algorithm," Fast Software Encryption, Second International Workshop Proceedings, Springer-Verlag, pp. 97-110.

[19] B. Schneier,1994, "Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish)," Fast Software Encryption, Cambridge Security Workshop Proceedings, Springer-Verlag, pp. 191-204.

[20] P. Rogaway and D. Coppersmith,1994, "A Software-Optimized Encryption Algorithm," Fast Software Encryption, Cambridge Security