University of Sudan for Science and Technology

College of graduate studies

College of Computer Science and Information Technology

Performance Evaluation of Blowfish Encryption Algorithm

تقييم أداء خوارزمية بلوفش (Blowfish) للتشفير

A Thesis submitted in partial fulfillment of the requirements for the Degree of Master of Information Technology

Prepared by:

Mohaned Abduallah Elshaikh

Supervised by:

Dr. Faisal Mohammed Abdallah

October 2017

الآية

بسم الله الرحمن الرحيم

"ربِّ قَدْ ءَاتَيْتَنِي مِنَ المُلْكِ وعَلَّمْتَنِي مِن تَأْوِيلِ الأَحَادِيثِ فَاطِرَ السَّماواتِ والأَرْضِ أَنتَ وَلِيِّ فِي الدّنيا والأَخِرَةِ تَوَفّنِي مُسْلِماً وَأَلْحِقنِي بِالصَّالِحِينَ "

صدق الله العظيم

سورة يوسف الآية (101)

Acknowledgement

First, I would like to thank Allah for giving me the power and health to do this work.

Second, I would like to express my special thanks to my supervisor **Dr: Faisal Mohammed Abdallah** for the guidance, encouragement and advice.

I am indebted to the Sudan University of Science &Technology for providing the facilities to conduct this work.

Finally, grateful acknowledgement is made to all those who participated with their time, effort, advise and knowledge to make this a successful study.

Abstract

The revolution and improvement of life in terms of technologies, allow users to share their data and information and this is the one of common uses of computer technologies. But with world full of third parties, who is capable of collect, theft and destroy information of others without any authority, and cryptography is method was developed to help users to keep their information secure using different algorithms. Blowfish is a cipher algorithm and it is simple, secure and efficient, and the performance of blowfish effected by the size of message and key length. The research aims to evaluate the performance of Blowfish by modifying the structure of F function. The modified Blowfish will use just two S-box in F function instead of four that used in Blowfish to compare encryption time and security. Encryption time and decryption time were calculated to compare between Blowfish and modified Blowfish and results state that the modified Blowfish is better in term of performance. The results of Diehard Battery that used to test the randomness shows that Blowfish algorithm is in high level security more that modified Blowfish, so is better to use Blowfish algorithm to encrypt data and applications that need a high level of security

المستخلص

ثورة تقنيات الحاسوب وغيرها من التقنيات في العصر الحديث، سمحت لمستخدميها بمشاركة البيانات الخاصة بهم وسهولة تداولها. لكن هنالك أطراف تسعى لجمع تلك البيانات واستخدامها من دون أي صلاحية من مالك هذه البيانات، والتشفير هو احدى التقنيات المستخدمة لحماية البيانات من السرقة او التدمير او التعديل، وهو تحويل الرسائل المرسلة بين المرسل و المتلقي الى صيغة غير مفهومة للآخرين. وواحدة من خوارزميات التشفير هي Blowfish وهي خوارزمية سهلة وبسيطة و فعالة ولكن أداءها يتأثر بحجم النص المراد تشفيره وطول المفتاح. يهدف البحث إلى تقييم أداء خوارزمية Blowfish عن طريق تعديل هيكلة (F-function). سيتم تعديل Blowfish باستخدام اثنين من S-box في وظيفة F-function بدلاً من أربعة تستخدم في Blowfish لمقارنة زمن وقوة التشفير. تم حساب وقت التشفير ووقت فك التشفير للمقارنة بين Blowfish و Blowfish المعدلة والنتائج تشير إلى أن Blowfish المعدلة أفضل من ناحية الاداء والزمن النستغرق في عملية التشفير. نتائج اختبارات Diehard Battery التي تستخدم لاختبار العشوائية تبين أن خوارزمية Blowfish تملك قوة تشفير أكثر من Blowfish المعدلة ولذلك من الافضل استخدام خوارزمية للتشفير خاصة عند تشفير بيانات أو تطبيقات تحتاج لمستوى أمن وحماية عالي.

Table of Contents

List of Tables

List of Figures

LIST OF ABBREVIATIONS

Abbreviation	Stand For
S-box	Substitution Box
P-array	Permutation Array
DES	Data Encryption Standard
AES	Advance encryption Standard
NIST	National Institute of Standards and Technology
STS	Statistical Test Suite

List of Appendices

Chapter 1

Introduction

1.1 Overview:

The exchanging information is one of the main uses of electronic communication. While cyber-crime increased, the securing of critical information is the most need nowadays.

There many tools used to maintain information security and Cryptography is one of these tools. Cryptography aims to transmit confidential information through insecure channels such as internet. The main goals of cryptography are authentication, confidentiality, integrity and non-repudiation. The message that needed to be transmitted using cryptography is called plaintext, and the coded message is called ciphertext. Converting plaintext to ciphertext is known as encryption and get the plaintext from ciphertext is decryption. [1]

The design of blowfish was in 1993 as high performance algorithm and to be used as freely alternate to encryption algorithms which designed earlier. There are frequently analysis on Blowfish, and this algorithm is not widely accepted as a robust encryption algorithm. Blowfish encryption has many positives. It is appropriate to work in hardware applications and it van be utilized without any fees. [4]

1.2 Problem Statement

In all of the symmetric key algorithms, Blowfish Encryption Algorithm is the most efficient one. Blowfish is designed to process a strong encryption algorithm so it has been used in many applications. But the variability of key length in Blowfish algorithm leads to low speed in encryption and decryption and high resources consumption.

1.3 Objectives

- Evaluate the performance in term of time needed to encrypt and decrypt the message.
- Assess the security of Blowfish Encryption.

1.4 Methodology

To evaluate the security of Blowfish encryption and performance, Blowfish algorithm will compare with modified Blowfish. Blowfish has three main sections S-Box preparation, sub keys generation and Encryption. The modification will be simple by changing the structure of F- function using less S-box.

1.5 Research scope:

Research activities in evaluate the performance and effectiveness of Blowfish algorithm when increasing the size of key.

In this research, the evaluation go through modifying the F function which is component of block cipher structured in Fiestel Network.

The areas that including in the research are

- Cryptography.
    - Block Cipher (Fiestel Network).

1.6 Thesis outline:

This research is organized as **5** chapters:

- Chapter (1) overview of the research topic and problem statement and research objectives.
- Chapter (2) Literature review and Related work.
- Chapter (3) Proposed Method and Tools.
- Chapter (4) Results and discussion.
- Chapter (5) Conclusion and Recommendation.

Chapter 2

LITERATURE REVIEW AND RELATED WORKS

2.1 Introduction

As the importance and the value of exchanged data over the internet or other media types are increasing, the search for the best solution to offer the necessary protection against the data thieves' attacks along with providing the services under timely manner is one of the most active subjects in the security related communities.

2.1.1 Cryptography

The origin of term cryptography is words Crypto and Graphy which are a Greek words. The meaning of Crypto and Graphy is secret and writing. The main objective of cryptography is generating messages that needed to be transmitted through insecure communications links. [11]

The converting from plaintext to ciphertext is known as encryption or enciphering. The retaining plaintext from ciphertext is called decryption or deciphering. [11]

The figure 2.1 describe the process of cryptography which usually referred to as "the study of secret". The figure describe the process of converting the message to ciphertext (encryption) and the reverse process (decryption). To complete the both processes (encryption and decryption) the known key must be used. [6]

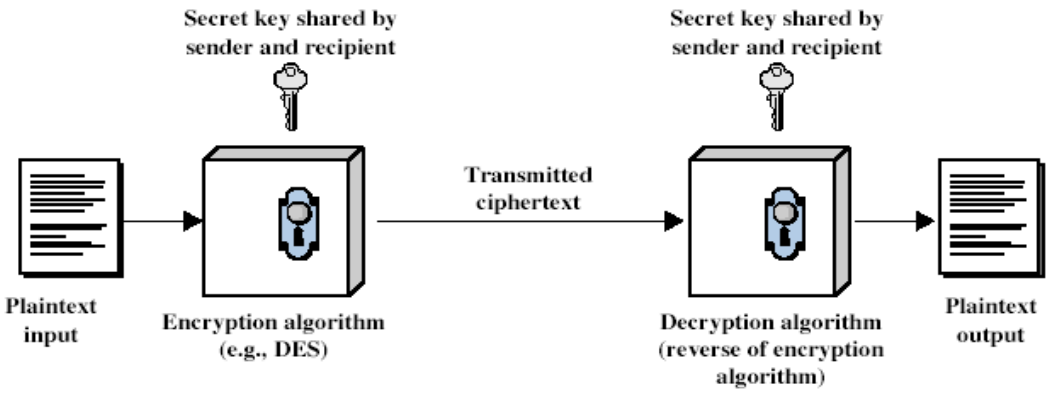

Figure 2.1 Encryption / Decryption Process [6]

2.1.2 Cryptography Goals

Each security system must implement a number of security function to maintain the secrecy of the system. Without these function the security system can't be called as "a security system":

1. CONFIDENTIALLY: data is available just by approved clients.

2. Confirmation: Origin of message is effectively related to an affirmation that character isn't false.

3. INTERGRITY: Only approved clients can change put away or transmitted data.

4. NON REPUDIATION: The capacity of preclude from claiming transmission message just by sender or collector.

5. ACCESS CONTROL: Managing the entrance to data or framework.

6. AVAILIBILITY: Make all assets accessible to approved clients when required. [6]

2.2 Classical Encryption Techniques

All classical encryption techniques are substitution or transposition.

- Substitution: replacing an element of the plaintext with other element.
- Transposition: changing the position of the elements of the plaintext.
- Transposition is also known as permutation.[4]

2.2.1 Caesar cipher

One of the least complex cases of a substitution cipher is the Caesar cipher. It is said to have been utilized by Julius Caesar to speak with his armed force. Caesar chose that moving each letter three places down the letter set in the message would be his standard calculation, thus he educated the majority of his commanders of his choice, and was then ready to send them encoded messages. One of the qualities of the Caesar figure is its usability and this convenience would be vital for Caesar since his warriors were likely uneducated and not fit for utilizing a confused coding framework. [8]

Facilitate improvement to unique three spots moving of character in Caesar cipher utilizes modulo twenty six number juggling for encryption key that is more noteworthy than 26:

$$Enx = x+nmod\ 26$$

The most squeezing shortcoming of this cipher is straightforwardness of its encryption and decryption calculations; the framework can be deciphered without knowing the encryption key. It is effortlessly broken by turning around encryption process with straightforward move of letter set requesting. [8]

2.2.1.1 Caesar cipher Limitations

1. It can't utilize unique character and numbers.
2. Space between two words in the plaintext isn't considered as one character.
3. All the 25 conceivable keys can be striven for the simple ID of the plaintext. [11]

2.2.2 Vigenere Cipher

This cipher when contrasted with Caesar gives some level of security with the presentation of a key; this key is rehashed to cover the length of the plaintext that will be scrambled illustration is demonstrated as follows:

KEY: f a u z a n f a u z a n

P.T: c r y p t o g r a p h y

Cipher: H R S O T B L R U O H L

As should be obvious from above case that "fauzan" is our watchword and plain content is "cryptography" which was scrambled in to "HRSOTBLRUOHL" this was finished utilizing Vigenere table which consest of letter sets in type of lines and segments left most segment demonstrates catchphrases and best most column shows message and at the intersection of two letters lives our substitution and after independently changing each letter we get an encoded message.[12]

2.2.3 Multiple Character Encryption to Mask Plain Text Structure

One character at any given moment substitution clearly leaves excessively of the message structure in figure content So what about crushing a portion of that structure by mapping various characters at an opportunity to figure content characters The best known method that does numerous character substitution is known as Playfair Cipher. [16]

Developing the Matrix for Pair Wise Substitutions in PlayFair Cipher In this cipher pick an encryption key At that point fill the characters of the keyword in the first cells of a 5 × 5 network in a left to right design beginning with the cell at the upper left corner Fill whatever is left of the matrix elements location with the remaining of the letters in alphabetic request The characters I and J are doled out a similar cell In the accompanying case the keyword is "MONARCHY".[16]

M	O	N	A	R
C	H	Y	B	D
E	F	G	I/J	K
L	P	Q	S	T
U	V	W	X	Z

Figure 2.2 Playfair Matrix

For the encryption procedure the Playfair utilizes a couple of basic guidelines identifying with where the character fall in matrix

1. If the two characters are in a similar column take the character to one side of every one the match of letters "hb" in plaintext will get supplanted by "YD" in figure content.
2. If the two letters are in a similar section take the letter underneath every one returning to the best if at the base The combine "cl" of plaintext will get supplanted by "EU" in figure content.

3. If neither of the first two tenets are valid frame square shape with the two letters and take the character on the even inverse corner of the square shape The match "ap" of plaintext will supplanted by "OS" in cipher content. [16]

2.2.3.1 Dealing with Duplicate Letters in a key and Repeating Letters in Plaintext

You should drop any copies in a key Prior to the substitution rules are connected you should embed a picked "Filler" letter suppose it is 'x' between any rehashing characters in the message.So a message word for example "yahoo" progresses toward becoming "hurxray". [16]

2.3. Cryptosystem

A cryptosystem is the arranged rundown of limited conceivable plaintexts limited conceivable figure content limited conceivable keys and the cipher algorithms. In this keys are essential, as figures without variable keys can be easily broken with just the information of the cipher utilized and are hence pointless for generally purposes.[10]

2.3.1 Asymmetric Key Cryptography

The Asymmetric Key Cryptography utilizes distinctive keys for enciphering and deciphering The encryption key is open with the goal that anybody can scramble a message In any case the unscrambling key is private with the goal that exclusive the beneficiary can decode the message Usually to set up "key sets" inside a system with the goal that every client has an open and private key The general population key is made accessible to everybody with the goal that they can deliver information yet the private key is just made accessible to the individual it has a place with. [11]

2.3.2 Symmetric key cryptography

In symmetric cipher, same key is utilized for encryption and unscrambling activity. The information which has been scrambled is called ciphertext through which a cipher is taken after to move the information into ciphertext. Decryption strategy is followed in the turnaround request to decipher the ciphertext into unique technique with the data of length of message. Thusly, in the field of scrambling and unscrambling the procedure of change of plaintext to ciphertext with the utilization of unique key happens. [9]

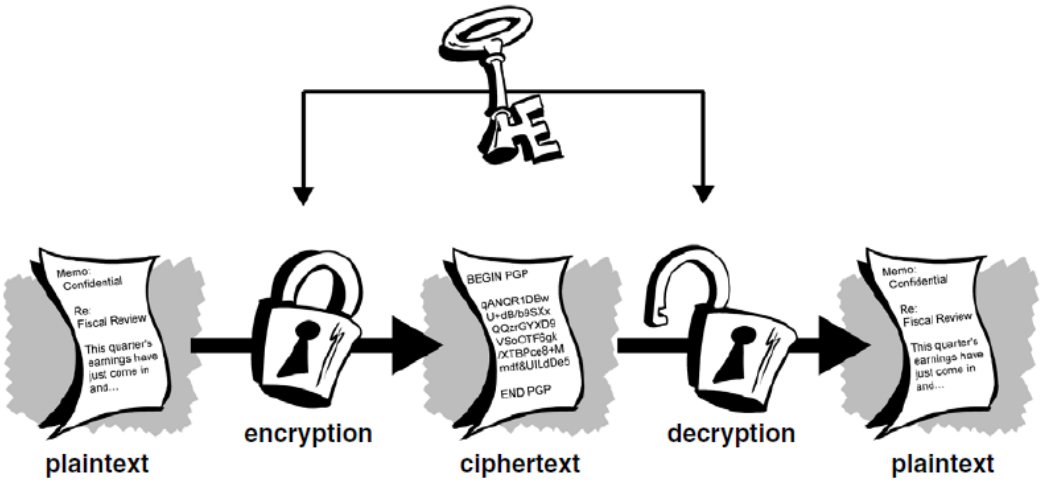

Figure 2.3 Symmetric key cryptography [13]

A symmetric encryption has five component:

1. **Plaintext:** The first message or information that is bolstered into the cipher as info.
2. **Encryption Algorithm:** performs different substitutions and changes on the plaintext.
3. **Secret Key:** The key is an esteem autonomous of the plaintext and of the cipher. The cipher will create an alternate yield contingent upon the particular key being utilized at the time.
4. **Ciphertext:** This is the mixed message created as yield It relies upon the plaintext and mystery key
5. **Decryption Algorithm:** This is basically the encryption calculation keep running backward. It takes the ciphertext and the mystery key and creates the first plaintext. [17]

2.3.3 Block Cipher and Stream Cipher

One of the fundamental classification strategies for encryption system usually utilized depends on the type of information they work on. The two sorts are Stream and Block Cipher.

2.3.3.1 Stream Cipher

Stream cipher works on a flood of information by working on it by bits. Stream algorithm comprises of two noteworthy segments a key stream generator and a blending capacity. Blending capacity is normally only a XOR work, while key stream generator is the fundamental part in stream algorithm encryption system. For instance, if the key stream generator makes arrangement of zeroes the yielded figured stream will be indistinguishable to the first plaintext. [6]

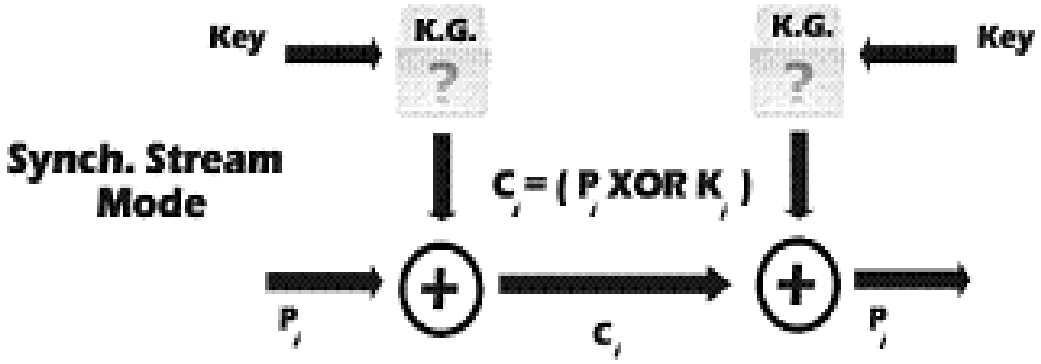

Figure 2.4 STREAM CIPHER (SIMPLE MODE) [6]

2.3.3.2 Block Cipher

In this technique information is enciphered and deciphered if information is in type of blocks. In its least complex mode, you isolate the plaintext into blocks which are bolstered into the figure framework to create blocks of ciphertext. ECB (Electronic Codebook Mode) is the fundamental type of block encryption where information blocks are enciphered straightforwardly to produce its comparing cipher blocks. [6]

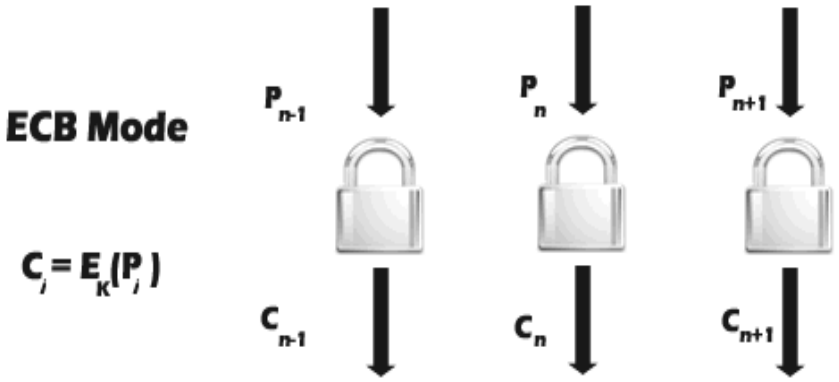

Figure 2.5 BLOCK CIPHER [6]

2.3.3.2.1 Feistel Networks

Feistel network is a general technique for changing any function (more often than not called a F-function) into a stage It was designed by Horst Feistel and has been utilized as a part of many Block encryption algorithms designs The flow of a Feistal design is as follow:

1. Split every block into equal parts.
2. Right half turns out to be new left part.
3. New right part is the last outcome when the left part is XOR'd with the aftereffect of applying f to the correct part and the key.
4. Note that past rounds can be inferred regardless of whether the capacity f isn't invertible. [13]

2.3.3.2.2 Substitution Boxes (s-boxes)

In the designing of symmetric cipher which are developed as substitution change (S-P) networks, a critical bit of the time taken on outline or on investigation is focused on the substitution boxes s boxes of the cipher. This is on the grounds that the rest of the cipher is linear severe shortcomings in the s-boxes can hence prompt a cipher which is effectively broken. [24]

There are a few properties which we feel to be cryptographically attractive in s-box. We can view these as our assessment criteria at that point and utilize them to manage our s-box plan. They are:

(1) Bijection;

(2) Nonlinearity;

(3) Strict avalanche;

(4) Independence of output bits.

Property (1) just guarantees that all conceivable 2n n bit input vectors will guide to unmistakable yield vectors K, this is a fundamental condition for invertibility of the s-box (which might possibly be imperative, contingent upon the framework of the encompassing system) and guarantees that all yield vectors show up once (which ensures that all conceivable information vectors are accessible to the following stage in the system). [24]

Property (2), guarantees that the s-box is definitely not a direct mapping from input vectors to yield vectors (since this would render the whole cryptosystem effortlessly flimsy). [24]

Property (3), the strict avalanche slide standard requires that for each information bit i rearranging bit i causes yield bit j to be modified half of the time (over all conceivable information vectors), for all j. Such s-boxes display what is for the most part alluded to as 'great avalanche," where rearranging any info bit i makes around half of the yield bits be reversed this is proportional to great "diffusion" and expands the property of "fulfillment" characterized in. [24]

Property (4), autonomy of yield bits guarantees that any two yield bits x and y act "freely" of each other; that is, bit x and y are not equivalent to each other essentially more, or fundamentally less, than a fraction of the time (over all conceivable input vectors). This is important since has indicated how the inquiry space can be lessened in specific attacks if the relationship between two yield bits is altogether other than zero.[24]

2.3.3.2.3 Data Encryption Standard (DES) Algorithm

DES is stand for Data Encryption Standard and it is first standard that recommended by NIST.IBM Team developed DES encryption around 1974 and approved as national standard in 1997. The block size is 64 bits and key length is under 56 bits key. DES operations are initial permutation, 16 rounds and final permutation. DES is used in many applications such as military and commercials. Although the DES standard is public, the design criteria used are classified. [17]

2.3.3.2.4 Advanced Encryption Standard (AES)

AES is a replacement of DES as NIST's new encryption standard. Earlier it was known as Rijndael (pronounced Rain Doll).It is a best encryption standard in 1997. The key length is variable (128, 192, or 256 bits) and 256 bits is default. Encryption using AES is flexible and fast. The only effective attack against AES is Brute force, when trying all characters combinations to break the cipher.[5]

2.3.3.2.3. Blowfish

Blowfish algorithm designed in 1993 by Brute Schneier Blowfish is a block cipher and it used as replace of DES or IDEA. The block size is 64 bits and the key length is from 42 up to 448 bits. [1]

2.3.3.2.4. Compare between AES, DES and Blowfish

Comparison between AES, DES and Blowfish was conducted in various encryption modes. Figure 2.6 shows the results of comparison in term of performance using different data block size. The figure shows that Blowfish is operate faster than other algorithms. When the block size is big, more resources will be consumed by AES.[22]

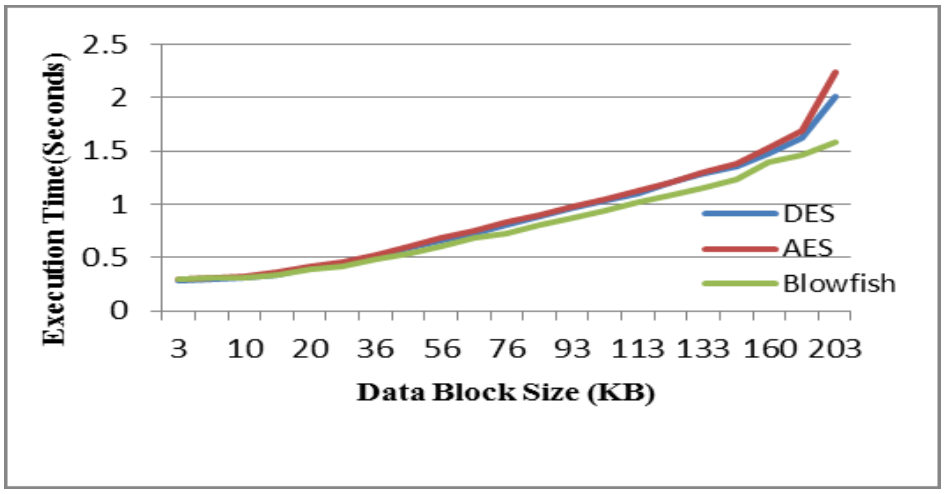

Figure 2.6 Comparison between AES, DES and Blowfish [22]

2.4 Previous Works

This part includes some approaches used to enhance Blowfish cipher by using different models.

2.4.1 Harsh K.Verma and Ravindra K. Singh (2012): they compared the performance of RC5, DES and Blowfish by providing some measurements on encryption. In their experiments, number of parameters affect the performance of above encryption algorithms such as: key length, number of rounds, block size and functions used in algorithm. The ciphers executed 10 times using different files types. According to execution time and resources utilization, RC5 is simpler and faster than DES and Blowfish block encryption.[2]

2.4.2 A.Ramesh and Dr.A.Suruliandi (2013): they analyzed and compared the performance of AES, DES and Blowfish. The performance of these algorithms evaluated based on execution time, throughput and memory required for implementation. To conduct the experiment, nine text files with different sizes used to obtain results. Based in results, Blowfish is 4 times faster than AES and 2 times faster than DES. Also Blowfish consumes less memory as compared with AES and DES. [3]

2.4.3 Diaa Salama, Hatem Mohamed and Mohie Mohamed (2008), in this study, the performance of symmetric encryption algorithms was evaluate. Experiments was conducted assess the metrics encryption time, CPU process time and CPU clock cycles and battery power for DES, 3DES, RC2, RC6, AES and Blowfish. The simulation results shows that the blowfish algorithm consumed less time for encryption operation when used text file. Also, this study checked the effect of packet size and file type on power consumption.[15]

2.4.4 Manikandan G, Rajendran P, Chakarapani K, Krishnan G and Sundarganesh G(2012): they aimed to enhance the data security by modifying Blowfish encryption. To improve the performance of blowfish algorithm, they modified the structure of F-function. Blowfish F-function is defined as follows: F(x)=((S1+S2 mod 232) XOR S3) + S4 mod 232. They Modified F-function by replacing 2 addition operations by XOR operations. The modified F-function become as: F(x)=((S1 XOR S2 mod 232) + (S3 XOR S4 mod 232)). The execution time results indicate that the modified Blowfish is faster than the original Blowfish without affect the security.[14]

2.5 Application Development:

2.5.1 C#.NET

C# is a modern and object programming language that have capabilities to easily and quickly solutions.

2.5.2 C# Main Features

1. SIMPLE

No pointers in C#, unsafe operations are not allowed and wide range of primitive types. [25]

2. MODERN

C# is following the current trend and is powerful for building robust applications. [25]

3. OBJECT ORIENTED

C# supports Data Encapsulation, polymorphism, inheritance, interfaces. C# enable the primitive types to become objects by using structures (struct). [25]

4. INTEROPERABILITY

C# includes native support for the COM and windows based applications.[25]

Chapter 3

Methodology

3.1 Blowfish Encryption

Information encryption happens by means of a 16-round Feistel arrange. Each round comprises of a key-subordinate change, and information subordinate substitution. All tasks are XORs and augmentations on 32-bit words. The main extra activities are four filed exhibit information queries per round. Blowfish utilizes countless. These keys must be pre-registered before any information encryption or unscrambling. The key cluster additionally called P-exhibit comprises of 18 32-bit sub-keys: P1, P2,...,P18.[18]

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.

Encryption: Blowfish is a Feistel organize comprising of 16 adjusts as in Figure 3.1. [18]

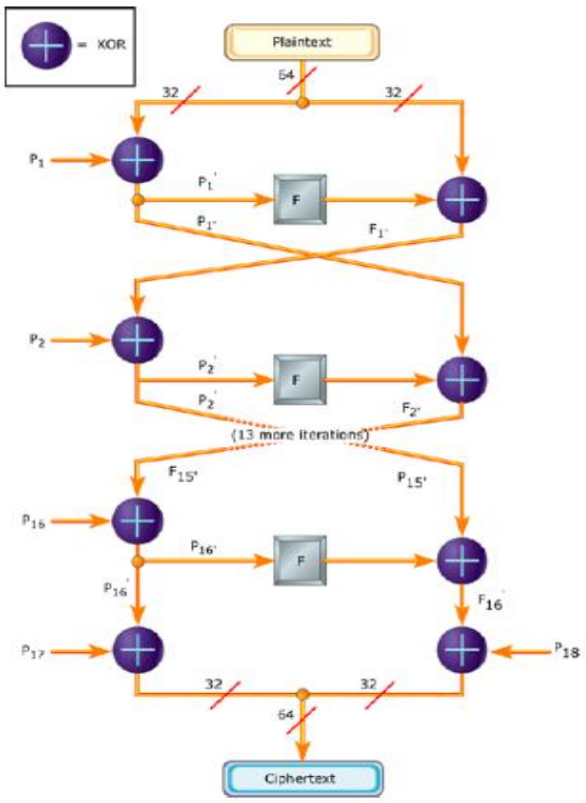

Figure 3.1 Blowfish algorithm [7]

The information is a 64-bit information component, x. Gap x into two 32-bit parts: xL, xR

For i= 1 to 16:

xL = xL XOR Pi

xR = F(xL) XOR xR

Swap xL and xR

Swap xL and xR (Undo the last swap.)

xR= xR XOR PI7

xL = xL XOR P 18

Recombine xL and xR

Function F:

Partition xL into four eight-piece quarters: a, b, c, and dF(xL) = ((SI,a + S2,b mod 232) XOR S3,c) +

S4,d mod 232

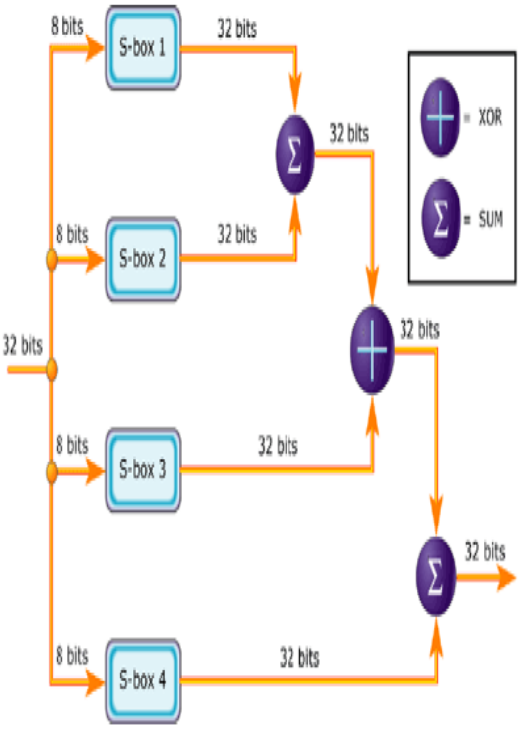

Figure 3.2 Graphic representation of Function F [7]

Decryption is precisely the same as encryption, with the exception of that PI, P2 ... P 18 are utilized as a part of the switch arrange. Executions of Blowfish that require the quickest speeds ought to unroll the circle and guarantee that all sub keys are put away in cache store.[7]

3.2 Methodology

The performance evaluation of Blowfish algorithms will be in two parts. The first evaluation is related to the time that encryption takes to convert the message (plaintext) to cipher text and the second evaluation will be process by test the security of Blowfish encryption. The comparison between Blowfish algorithm and modified blowfish will drive to results of performance evaluation.

3.2.1 Modified Blowfish

They optimize Blowfish by modifying the structure of F-function and no change in the Feistel structure of Blowfish algorithm. The optimized Blowfish uses two S-boxes in F-function rather than four S-box that used in F-function of Blowfish. Figure 3.3 display the modified structure of F-function. [26]

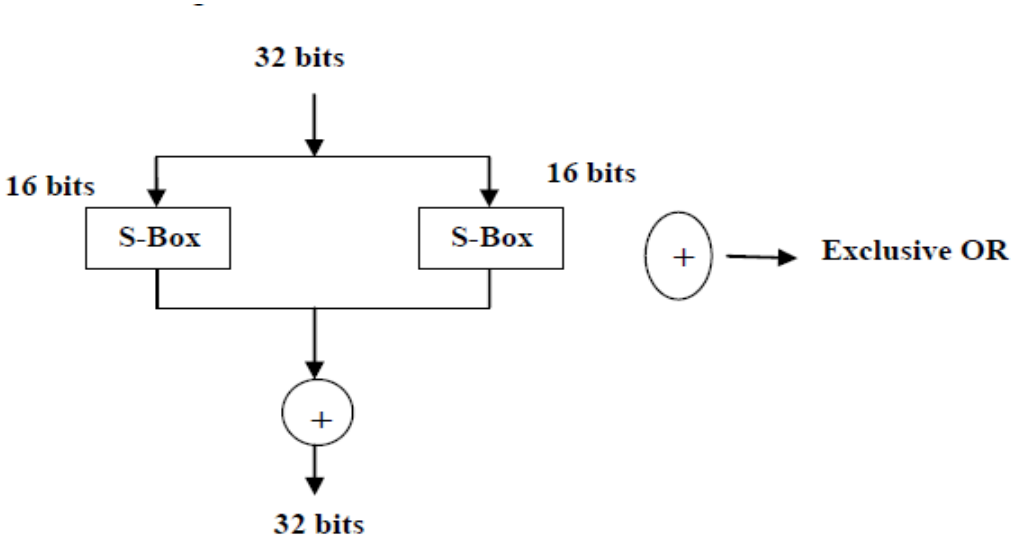

Figure 3.3 Modified F-function [26]

3.2.1.1 Pseudo code

A. Pseudo-code of F-Function with four S-Boxes (S0, S1, S2 and S3)

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

2: F(xL)=((S0,a+S1,b mod 2_{32})^S2,c)+S3,d mod 2_{32}.

B. The Pseudo-code of optimized F function with two S-boxes

1: Divide xL into two sixteen-bit quarters: a, and b.

2: F(xR)=(S0,a^ S1,b)

C. Pseudo-code of Encryption

1: Divide the 64 bit input data into two 32-bit halves (left and right): xL and xR

2: for i=0 to16

xL is XORed with P[i].

Find F(xL)

F(xL) is XORed with xR.

Interchange xL and xR.

3: Interchange xL and xR.

4: xR is XORed with P[16].

5: xL is XORed with P[17].

6: Finally combine xL and xR.

D. Pseudo-code of Decryption

1: Divide the 64 bit input data into two 32-bit halves (left and right): xL and xR

2: for i=17 to1

xL is XORed with P[i].

Find F(xL)

F(xL) is XORed with xR.

Interchange xL and xR.

3: Interchange xL and xR.

4: xR is XORed with P[1].

5: xL is XORed with P[0].

6: Finally combine xL and xR [26]

3.2.2 Performance Analysis and Comparison

3.2.2.1 Time Comparison

The Modified Blowfish algorithm will be compared with the Blowfish by calculating number of parameters. The first parameter is encryption time that needed to encrypt the plaintext. The second parameter is the decryption time which is needed to decrypt the ciphertext. The third parameter is total of encryption time and decryption time which is called execution time. After calculating these parameters, the difference between Blowfish algorithm and Modified Blowfish will show which algorithm performs better. [26]

The other parameter can be used to evaluate the performance of Blowfish and Modified Blowfish is a throughput. Throughput indicates the time that encryption need to convert plaintext to ciphertext. Throughput is calculated by this formula: Throughput=Total size of plaintext /Total execution time (Bytes used to measure plaintext in and milliseconds for total execution time). [26]

3.2.2.2 Randomness

Statistical randomness tests are capacities that take discretionary length information and create a real number somewhere in the range of 0 and 1 called the p value, which is delivered by assessing certain irregularity properties of the known information. For instance, Frequency Test generates p values relying upon the quantity of ones of every a paired succession, where Overlapping Template Test assesses the quantity of events of a particular arrangement of bits. [19]

3.2.2.3 Random Number Generation Tests

The NIST Test Suite is a measurable bundle comprising of 15 tests that were created to test the randomness of (arbitary long) binary groupings delivered by either equipment or programming based cryptographic random. These tests center around a wide range of sorts of non-haphazardness that could found in an arrangement. [20]

3.2.2.4 The Frequency (within block) Test

The focal point of the exam is the extent of ones inside M-bit blocks. The reason for this test is to decide if the recurrence of ones of every N-bit block is roughly N/2, as would be normal under a supposition of randomness. [24]

3.2.2.5 Non-overlapping Template Matching Test:

The focus of this test is the number of occurrences of pre-specified target strings. The purpose of this test is to detect generators that produce too many occurrences of a given non-periodic (aperiodic) pattern.[24]

The focal point of this exam is the quantity of events of pre-indicated target strings. The reason for this test is to identify generators that create excessively numerous events of a specified aperiodic design. [24]

3.2.2.6 Diehard Test

The quality of random number generator can be measured by the statistical test of the Diehard Battery They were developed by George Marsaglia over several years and published in 1995.[21]

3.2.2.6.1 Overlapping Sums Test

Create a long series of random floats on 0 1 Add series of 100 sequential floats characteristic sigma and mean should be used to distribute the sums. [21]

3.2.2.6.2 Runs

Create a long series of random floats on (0, 1). Count descending and ascending runs. A specific distribution should be used for the counts. [21]

3.2.2.6.3 Count the 1s

Count the 1 bits in each of either chosen bytes or successive. Covert 1s to “characters” then count the number of five characters “words”. [21]

Chapter 4

RESULTS AND DISCUSSION

This part includes the application used in encryption and decryption text using Blowfish and modified Blowfish and the results of randomness tests.

4.1 Implementation Environment

4.1.1. Application Interface

The below images display the interface that allow users to encrypt/ decrypt a text using Blowfish/Modified Blowfish.

In figure 4.1, encryption/decryption of small size of text. The interface have two sections, the first one for Blowfish ant the other for modified Blowfish. There are two controls to inter the message/ciphertext and key and select the operation you need to process. When the button Encrypt/Decrypt pressed the ciphertext/message will display in a label and the time token by the operation encryption/decryption will appear also.

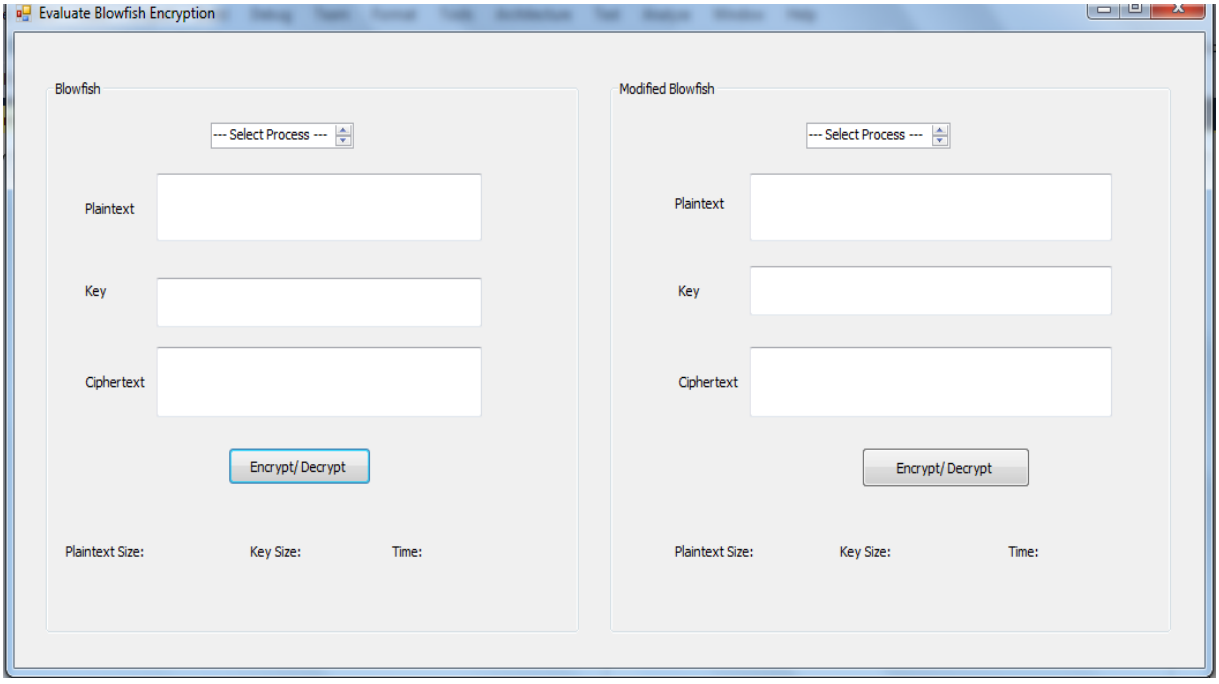

Figure 4.1 Encrypt/Decrypt text using Blowfish/Modified Blowfish

The application interface in figure 4.2 is used to encrypt a text file. The output file will be used in randomness tests and the size of it should be more than 10mb which needed by some tests to obtain results.

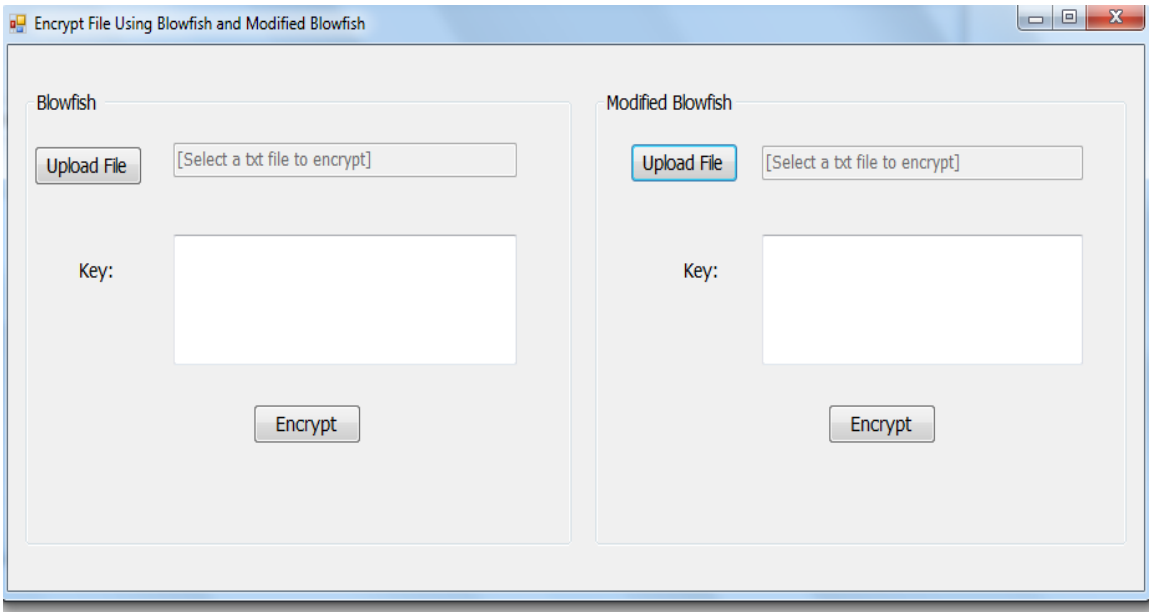

Figure 4.2 Encrypt text file using Blowfish/Modified Blowfish

4.2. Performance Test

The comparison is conducted only for the text. The program is executed 10 times to improve the accuracy of the timing measurement. The encryption time and decryption time is measured by milliseconds.

4.2.1. Performance Comparison based on Execution time

The encryption and decryption is conducted just for text. Time for encryption and decryption processes is calculated for different palintext size (in Bytes) and the key size is various also in 10 times. The execution time is the summation of encryption time and decryption time. The execution time and an average of it has been calculated.

Table 4.1 Comparison based on execution time

Plaintext Size (Bytes)	Key Size (Bytes)	Blowfish		Modified Blowfish		Blowfish	Modified Blowfish
		Encryption Time	Decryption Time	Encryption Time	Decryption Time		
40	8	5.81	5.82	4.25	4.28	11.83	8.53
84	12	5.93	5.95	4.33	4.37	11.85	8.70
52	16	5.84	5.88	4.25	4.30	11.72	8.55
72	20	5.92	5.96	4.31	4.36	11.88	8.67
48	24	5.84	5.89	4.28	4.30	11.73	8.58
72	28	5.93	5.94	4.35	4.35	11.87	8.70
64	36	5.90	5.93	4.31	4.35	11.83	8.66
56	40	5.85	5.91	4.30	4.31	11.76	8.61
52	44	5.87	5.92	4.31	4.33	11.79	8.64
64	56	5.92	5.93	4.32	4.34	11.85	8.66
Average Execution Time						11.81	8.63

The above table displays the results of comparison between Blowfish and Modified Blowfish in term of performance. The results shows that the Modified Blowfish is better depend on the average of execution time (Average of execution time is 8.63 for Modified Blowfish and 11.81 for Blowfish).

4.2.2. Performance Comparison based on Throughput

The Graph in Figure 4.3 shows the results of comparison based on throughput using different size of plaintext. As in graph throughput is high for Modified Blowfish. High throughput means less time for encryption process.

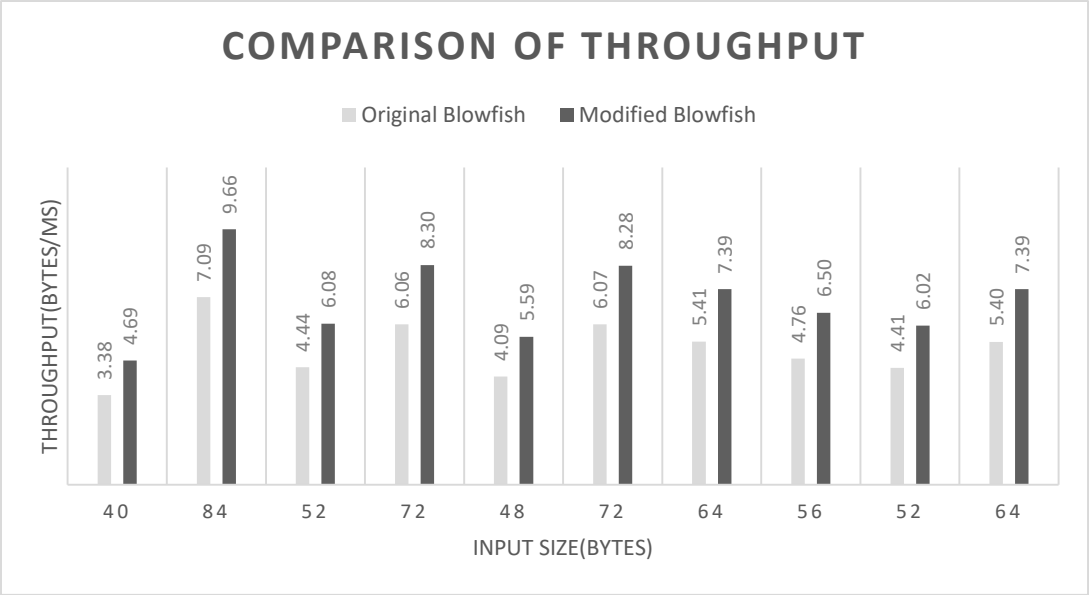

Figure 4.3 Comparison based on throughput

4.3. Randomness Tests

The randomness of algorithms is measured using STS tool, the result of the test is shown in the following tables:

Table 4.2 Results of STS tests between Blowfish and Modified Blowfish.

Test	Blowfish Test result	Status	Modified Blowfish Test result	Status
Block Frequency	p-value =1.000000	Passed	p-value =1.000000	Passed
Non overlapping Template matching	p-value =1.000000	Passed	p-value =1.000000	Passed

The above table show the result standard Blowfish and enhanced Blowfish algorithm after doing frequency test None overlapping Template matching by using STS tool.

4.4. Diehard Test

Also the randomness of algorithm is measured using diehard testing tool, the result of test is shown in the following table.

Table 4.3 Results of Diehard tests of Blowfish and Modified Blowfish.

Test	Blowfish	Status	Modified Blowfish	Status
OVERLAPPING SUMS	p-value =1.000000	Passed	p-value =1.000000	Passed
COUNT-THE-1's TEST for specific bytes	Byte1 Chi square= 1697.983 Byte 8 Chi square= 1316.786	Passed	Byte1 Chi square= 32643.450 Byte 8 Chi square= 6949.088	Passed
Runs Second time Run time	runs up; ks test for 10 p's:.531201 runs down; ks test for 10 p's:.531201 runs up; ks test for 10 p's:. 821329 runs down; ks test for 10 p's:. 525476	Passed	runs up; ks test for 10 p's: 1.000000 runs down; ks test for 10 p's:1.000000 runs up; ks test for 10 p's: 1.000000 runs down; ks test for 10 p's:1.000000	Failed

The passed tests indicates that both Blowfish and Modified Blowfish has a good randomness and the p-value is acceptable. In count-the-1's test for specific bytes, the sample size was 256,000 and p-value is 1.0000000 for both algorithms, but the different in Chi square, in Modified Blowfish result Byte1 Chi square=32643.450 is greater than result in Blowfish with value= 1697.983 that is mean the number of 1’s in specific byte in Modified Blowfish is more randomness than Blowfish.

The result of runs test should be a float value, and the result of Modified Blowfish is 1.0000000 which means the test is failed for it and passed for Blowfish.

Chapter 5

CONCLUSION AND RECOMMENDATION

5.1 Conclusion:

Blowfish is one of the most effective cryptographic algorithm, but it has slow performance when the encrypting a large size of text. This research modified the F function structure and using 2 S-box rather than 4 in original Blowfish.

According to results of calculating the time that consumed to encrypt and decrypt a text and then calculating the execution time and throughput indicates that the modified Blowfish is better than Blowfish in term of performance, But the Blowfish makes encrypted data with a high level of security than modified Blowfish depend on the Diehard test results.

5.2 Recommendation:

For better security, it is highly recommended to use Blowfish encryption algorithm rather than using modified Blowfish, especially when encrypt sensitive data and applications.

For future purpose, the modified Blowfish can be test with other data type such as image, audio and video.

REFERENCES

[1]Agrawal, M., & Mishra, P. (2012). A Modified Approach for Symmetric Key Cryptography Based on Blowfish Algorithm. *International Journal of Engineering and Advanced Technology (IJEAT)*, *1*(6), 79–83.

[2] Harsh K. Verma, & Ravindra K. Singh. (2012). Performance Analysis of RC5, Blowfish and DES Block Cipher Algorithms. *International Journal of Computer Applications*, 42(16), 8-14.

[3] A.Ramesh & Dr.A.Suruliandi. (2013). Performance Analysis of Encryption Algorithms for Information Security. *International Conference on Circuits, Power and Computing Technologies*, 840-844.

[4] Singh, P., & Singh, P. K. (2013). IMAGE ENCRYPTION AND DECRYPTION. *International Journal of Scientific & Engineering Research*, *4*(7), 150–154.

[5] Singh, S. P., & Maini, R. (2011). COMPARISON OF DATA ENCRYPTION. *International Journal of Computer Science and Communication*, *2*(1), 125–127.

[6] Verma, O. P., Agarwal, R., Dafouti, D., & Tyagi, S. (2011). Performance Analysis of Data Encryption Algorithms. *IEEE*, *3*(11), 399–403.

[7] Yadav, R., & Joshi, T. (2015). Cryptography with merged encryption & decryption in VHDL. *International Journal of Research in Engineering Science and Technologies (IJRESTs)*, *1*(2), 19–23.

[8] Abraham, O. (2012). "AN IMPROVED CAESAR CIPHER (ICC) ALGORITHM." *International Journal Of Engineering Science & Advanced Technology*.

[9] Asiya Abdus Salam, R. M. A. S. (2015). "Vertically Scrambled Caesar Cipher Method." *International Journal of Computer Applications*.

[10] Krishna Kumar Pandey, V. R., Sitesh KumarSinha (2013). "An Enhanced Symmetric Key Cryptography Algorithm to Improve Data Security". *International Journal of Computer Applications*.

[11] Mohammed, B. B. (2013). "Automatic Key Generation of Caesar Cipher". *International Journal of Engineering Trends and Technology (IJETT)*.

[12] Saeed, F., & Rashid, M. (2010). Integrating Classical Encryption with Modern Technique. *International Journal of Computer Science and Network Security*, *10*(5), 280–285.

[13] B. Schneier, Applied Cryptography,John Wiley & Sons, New York, 1994.

[14] Manikandan G, Rajendran P, Chakarapani K, Krishnan G and Sundarganesh G (2012). "A Modified Crypto Scheme for Enhancing Data Security". *Journal of Theoretical and Applied Information Technology*, 35(2), 149-154.

[15] Diaa Salama, Hatem Mohamed and Mohie Mohamed. (2008). "Performance Evaluation of Symmetric Encryption Algorithm". *International Journal of Computer Science and Network Security, 8(12), 280-286.*

[16] Sharma, R. (2012). Classical Encryption Techniques. *International Journal of Computers & Technology*, *3*(1), 84–90.

[17] Singh, P., & Singh, P. K. (2013). IMAGE ENCRYPTION AND DECRYPTION. *International Journal of Scientific & Engineering Research*, *4*(7), 150–154.

[18] Krishnamurthy, G. N., Ramaswamy, V., Leela, G. H., & Ashalatha, M. E. (2008). Performance enhancement of Blowfish and CAST-128 algorithms and Security analysis of improved Blowfish algorithm using Avalanche effect. *International Journal of Computer Science and Network Security*, *8*(3), 244–250.

[19] Rukhin, A., et al. (2001). A statistical test suite for random and pseudorandom number generators for cryptographic applications, *DTIC Document.*

[20**]** Zaman, J. and R. Ghosh (2012). A review study of NIST Statistical Test Suite: Development of an indigenous computer package." arXiv preprint arXiv: 1208.5740.

[21]Alani, M. M. (2010). Testing Randomness in Ciphertext of Block-Ciphers Using DieHard Tests. *International Journal of Computer Science and Network Security*, *10*(4), 53–57.

[22] Thakur, J. and Kumar (2011) 'DES, AES and Blowfish: Symmetric key cryptography algorithms simulation based performance analysis', *International journal of emerging technology and advanced engineering*, 1(2), 6–12.

[23]Adams, C. and Tavares, S. (1990) 'The structured design of cryptographically good s-boxes', Journal of Cryptology, 3(1), 27–41.

[24] https://csrc.nist.gov/Projects/Random-Bit-Generation/Documentation-and-Software/Guide-to-the-Statistical-Tests [Access on 10/10/2017]

[25] https://www.c-sharpcorner.com/article/C-Sharp-and-its-features/ [Access on 15/10/2017]

[26] Christina L, Joe Irudayaraj V S (2014). Optimized Blowfish Encryption Technique, *International Journal of Innovative Research in Computer and Communication Engineering*, 2(7), 5009 – 5015.

Appendix A

1. Blowfish algorithm Code
   The C# code of Blowfish Encryption

```
using System;
using System.IO;
using System.Text;

namespace Simias.Encryption
{
    /// <summary>
    /// Class that provides blowfish encryption.
    /// </summary>
    public class Blowfish
    {
        const int N = 16;
        const int KEYBYTES = 8;

        static uint[] _P =
              {
                    0x243f6a88, 0x85a308d3, 0x13198a2e, 0x03707344,
0xa4093822, 0x299f31d0,
                    0x082efa98, 0xec4e6c89, 0x452821e6, 0x38d01377,
0xbe5466cf, 0x34e90c6c,
                    0xc0ac29b7, 0xc97c50dd, 0x3f84d5b5, 0xb5470917,
0x9216d5d9, 0x8979fb1b
               };
        static uint[,] _S =
              {

                           {0xd1310ba6, 0x98dfb5ac, 0x2ffd72db, 0xd01adfb7,
      0xb8e1afed, 0x6a267e96,
                                0xba7c9045, 0xf12c7f99, 0x24a19947,
      0xb3916cf7, 0x0801f2e2, 0x858efc16,
                                0x636920d8, 0x71574e69, 0xa458fea3,
      0xf4933d7e, 0x0d95748f, 0x728eb658,
                                0x718bcd58, 0x82154aee, 0x7b54a41d,
      0xc25a59b5, 0x9c30d539, 0x2af26013,
                                0xc5d1b023, 0x286085f0, 0xca417918,
      0xb8db38ef, 0x8e79dcb0, 0x603a180e,
                                0x6c9e0e8b, 0xb01e8a3e, 0xd71577c1,
      0xbd314b27, 0x78af2fda, 0x55605c60,
                                0xe65525f3, 0xaa55ab94, 0x57489862,
      0x63e81440, 0x55ca396a, 0x2aab10b6,
                                0xb4cc5c34, 0x1141e8ce, 0xa15486af,
      0x7c72e993, 0xb3ee1411, 0x636fbc2a,
                                0x2ba9c55d, 0x741831f6, 0xce5c3e16,
      0x9b87931e, 0xafd6ba33, 0x6c24cf5c,
                                0x7a325381, 0x28958677, 0x3b8f4898,
      0x6b4bb9af, 0xc4bfe81b, 0x66282193,
                                0x61d809cc, 0xfb21a991, 0x487cac60,
      0x5dec8032, 0xef845d5d, 0xe98575b1,
```

```
                            0xdc262302, 0xeb651b88, 0x23893e81,
0xd396acc5, 0x0f6d6ff3, 0x83f44239,
                            0x2e0b4482, 0xa4842004, 0x69c8f04a,
0x9e1f9b5e, 0x21c66842, 0xf6e96c9a,
                            0x670c9c61, 0xabd388f0, 0x6a51a0d2,
0xd8542f68, 0x960fa728, 0xab5133a3,
                            0x6eef0b6c, 0x137a3be4, 0xba3bf050,
0x7efb2a98, 0xa1f1651d, 0x39af0176,
                            0x66ca593e, 0x82430e88, 0x8cee8619,
0x456f9fb4, 0x7d84a5c3, 0x3b8b5ebe,
                            0xe06f75d8, 0x85c12073, 0x401a449f,
0x56c16aa6, 0x4ed3aa62, 0x363f7706,
                            0x1bfedf72, 0x429b023d, 0x37d0d724,
0xd00a1248, 0xdb0fead3, 0x49f1c09b,
                            0x075372c9, 0x80991b7b, 0x25d479d8,
0xf6e8def7, 0xe3fe501a, 0xb6794c3b,
                            0x976ce0bd, 0x04c006ba, 0xc1a94fb6,
0x409f60c4, 0x5e5c9ec2, 0x196a2463,
                            0x68fb6faf, 0x3e6c53b5, 0x1339b2eb,
0x3b52ec6f, 0x6dfc511f, 0x9b30952c,
                            0xcc814544, 0xaf5ebd09, 0xbee3d004,
0xde334afd, 0x660f2807, 0x192e4bb3,
                            0xc0cba857, 0x45c8740f, 0xd20b5f39,
0xb9d3fbdb, 0x5579c0bd, 0x1a60320a,
                            0xd6a100c6, 0x402c7279, 0x679f25fe,
0xfb1fa3cc, 0x8ea5e9f8, 0xdb3222f8,
                            0x3c7516df, 0xfd616b15, 0x2f501ec8,
0xad0552ab, 0x323db5fa, 0xfd238760,
                            0x53317b48, 0x3e00df82, 0x9e5c57bb,
0xca6f8ca0, 0x1a87562e, 0xdf1769db,
                            0xd542a8f6, 0x287effc3, 0xac6732c6,
0x8c4f5573, 0x695b27b0, 0xbbca58c8,
                            0xe1ffa35d, 0xb8f011a0, 0x10fa3d98,
0xfd2183b8, 0x4afcb56c, 0x2dd1d35b,
                            0x9a53e479, 0xb6f84565, 0xd28e49bc,
0x4bfb9790, 0xe1ddf2da, 0xa4cb7e33,
                            0x62fb1341, 0xcee4c6e8, 0xef20cada,
0x36774c01, 0xd07e9efe, 0x2bf11fb4,
                            0x95dbda4d, 0xae909198, 0xeaad8e71,
0x6b93d5a0, 0xd08ed1d0, 0xafc725e0,
                            0x8e3c5b2f, 0x8e7594b7, 0x8ff6e2fb,
0xf2122b64, 0x8888b812, 0x900df01c,
                            0x4fad5ea0, 0x688fc31c, 0xd1cff191,
0xb3a8c1ad, 0x2f2f2218, 0xbe0e1777,
                            0xea752dfe, 0x8b021fa1, 0xe5a0cc0f,
0xb56f74e8, 0x18acf3d6, 0xce89e299,
                            0xb4a84fe0, 0xfd13e0b7, 0x7cc43b81,
0xd2ada8d9, 0x165fa266, 0x80957705,
                            0x93cc7314, 0x211a1477, 0xe6ad2065,
0x77b5fa86, 0xc75442f5, 0xfb9d35cf,
                            0xebcdaf0c, 0x7b3e89a0, 0xd6411bd3,
0xae1e7e49, 0x00250e2d, 0x2071b35e,
                            0x226800bb, 0x57b8e0af, 0x2464369b,
0xf009b91e, 0x5563911d, 0x59dfa6aa,
```

```
                            0x78c14389, 0xd95a537f, 0x207d5ba2,
0x02e5b9c5, 0x83260376, 0x6295cfa9,
                            0x11c81968, 0x4e734a41, 0xb3472dca,
0x7b14a94a, 0x1b510052, 0x9a532915,
                            0xd60f573f, 0xbc9bc6e4, 0x2b60a476,
0x81e67400, 0x08ba6fb5, 0x571be91f,
                            0xf296ec6b, 0x2a0dd915, 0xb6636521,
0xe7b9f9b6, 0xff34052e, 0xc5855664,
                            0x53b02d5d, 0xa99f8fa1, 0x08ba4799,
0x6e85076a
                    },
                    {
                            0x4b7a70e9, 0xb5b32944, 0xdb75092e,
0xc4192623, 0xad6ea6b0, 0x49a7df7d,
                            0x9cee60b8, 0x8fedb266, 0xecaa8c71,
0x699a17ff, 0x5664526c, 0xc2b19ee1,
                            0x193602a5, 0x75094c29, 0xa0591340,
0xe4183a3e, 0x3f54989a, 0x5b429d65,
                            0x6b8fe4d6, 0x99f73fd6, 0xa1d29c07,
0xefe830f5, 0x4d2d38e6, 0xf0255dc1,
                            0x4cdd2086, 0x8470eb26, 0x6382e9c6,
0x021ecc5e, 0x09686b3f, 0x3ebaefc9,
                            0x3c971814, 0x6b6a70a1, 0x687f3584,
0x52a0e286, 0xb79c5305, 0xaa500737,
                            0x3e07841c, 0x7fdeae5c, 0x8e7d44ec,
0x5716f2b8, 0xb03ada37, 0xf0500c0d,
                            0xf01c1f04, 0x0200b3ff, 0xae0cf51a,
0x3cb574b2, 0x25837a58, 0xdc0921bd,
                            0xd19113f9, 0x7ca92ff6, 0x94324773,
0x22f54701, 0x3ae5e581, 0x37c2dadc,
                            0xc8b57634, 0x9af3dda7, 0xa9446146,
0x0fd0030e, 0xecc8c73e, 0xa4751e41,
                            0xe238cd99, 0x3bea0e2f, 0x3280bba1,
0x183eb331, 0x4e548b38, 0x4f6db908,
                            0x6f420d03, 0xf60a04bf, 0x2cb81290,
0x24977c79, 0x5679b072, 0xbcaf89af,
                            0xde9a771f, 0xd9930810, 0xb38bae12,
0xdccf3f2e, 0x5512721f, 0x2e6b7124,
                            0x501adde6, 0x9f84cd87, 0x7a584718,
0x7408da17, 0xbc9f9abc, 0xe94b7d8c,
                            0xec7aec3a, 0xdb851dfa, 0x63094366,
0xc464c3d2, 0xef1c1847, 0x3215d908,
                            0xdd433b37, 0x24c2ba16, 0x12a14d43,
0x2a65c451, 0x50940002, 0x133ae4dd,
                            0x71dff89e, 0x10314e55, 0x81ac77d6,
0x5f11199b, 0x043556f1, 0xd7a3c76b,
                            0x3c11183b, 0x5924a509, 0xf28fe6ed,
0x97f1fbfa, 0x9ebabf2c, 0x1e153c6e,
                            0x86e34570, 0xeae96fb1, 0x860e5e0a,
0x5a3e2ab3, 0x771fe71c, 0x4e3d06fa,
                            0x2965dcb9, 0x99e71d0f, 0x803e89d6,
0x5266c825, 0x2e4cc978, 0x9c10b36a,
                            0xc6150eba, 0x94e2ea78, 0xa5fc3c53,
0x1e0a2df4, 0xf2f74ea7, 0x361d2b3d,
```

```
                            0x1939260f, 0x19c27960, 0x5223a708,
0xf71312b6, 0xebadfe6e, 0xeac31f66,
                            0xe3bc4595, 0xa67bc883, 0xb17f37d1,
0x018cff28, 0xc332ddef, 0xbe6c5aa5,
                            0x65582185, 0x68ab9802, 0xeecea50f,
0xdb2f953b, 0x2aef7dad, 0x5b6e2f84,
                            0x1521b628, 0x29076170, 0xecdd4775,
0x619f1510, 0x13cca830, 0xeb61bd96,
                            0x0334fe1e, 0xaa0363cf, 0xb5735c90,
0x4c70a239, 0xd59e9e0b, 0xcbaade14,
                            0xeecc86bc, 0x60622ca7, 0x9cab5cab,
0xb2f3846e, 0x648b1eaf, 0x19bdf0ca,
                            0xa02369b9, 0x655abb50, 0x40685a32,
0x3c2ab4b3, 0x319ee9d5, 0xc021b8f7,
                            0x9b540b19, 0x875fa099, 0x95f7997e,
0x623d7da8, 0xf837889a, 0x97e32d77,
                            0x11ed935f, 0x16681281, 0x0e358829,
0xc7e61fd6, 0x96dedfa1, 0x7858ba99,
                            0x57f584a5, 0x1b227263, 0x9b83c3ff,
0x1ac24696, 0xcdb30aeb, 0x532e3054,
                            0x8fd948e4, 0x6dbc3128, 0x58ebf2ef,
0x34c6ffea, 0xfe28ed61, 0xee7c3c73,
                            0x5d4a14d9, 0xe864b7e3, 0x42105d14,
0x203e13e0, 0x45eee2b6, 0xa3aaabea,
                            0xdb6c4f15, 0xfacb4fd0, 0xc742f442,
0xef6abbb5, 0x654f3b1d, 0x41cd2105,
                            0xd81e799e, 0x86854dc7, 0xe44b476a,
0x3d816250, 0xcf62a1f2, 0x5b8d2646,
                            0xfc8883a0, 0xc1c7b6a3, 0x7f1524c3,
0x69cb7492, 0x47848a0b, 0x5692b285,
                            0x095bbf00, 0xad19489d, 0x1462b174,
0x23820e00, 0x58428d2a, 0x0c55f5ea,
                            0x1dadf43e, 0x233f7061, 0x3372f092,
0x8d937e41, 0xd65fecf1, 0x6c223bdb,
                            0x7cde3759, 0xcbee7460, 0x4085f2a7,
0xce77326e, 0xa6078084, 0x19f8509e,
                            0xe8efd855, 0x61d99735, 0xa969a7aa,
0xc50c06c2, 0x5a04abfc, 0x800bcadc,
                            0x9e447a2e, 0xc3453484, 0xfdd56705,
0x0e1e9ec9, 0xdb73dbd3, 0x105588cd,
                            0x675fda79, 0xe3674340, 0xc5c43465,
0x713e38d8, 0x3d28f89e, 0xf16dff20,
                            0x153e21e7, 0x8fb03d4a, 0xe6e39f2b,
0xdb83adf7
                    },
                    {
                            0xe93d5a68, 0x948140f7, 0xf64c261c,
0x94692934, 0x411520f7, 0x7602d4f7,
                            0xbcf46b2e, 0xd4a20068, 0xd4082471,
0x3320f46a, 0x43b7d4b7, 0x500061af,
                            0x1e39f62e, 0x97244546, 0x14214f74,
0xbf8b8840, 0x4d95fc1d, 0x96b591af,
                            0x70f4ddd3, 0x66a02f45, 0xbfbc09ec,
0x03bd9785, 0x7fac6dd0, 0x31cb8504,
```

```
                            0x96eb27b3, 0x55fd3941, 0xda2547e6,
0xabca0a9a, 0x28507825, 0x530429f4,
                            0x0a2c86da, 0xe9b66dfb, 0x68dc1462,
0xd7486900, 0x680ec0a4, 0x27a18dee,
                            0x4f3ffea2, 0xe887ad8c, 0xb58ce006,
0x7af4d6b6, 0xaace1e7c, 0xd3375fec,
                            0xce78a399, 0x406b2a42, 0x20fe9e35,
0xd9f385b9, 0xee39d7ab, 0x3b124e8b,
                            0x1dc9faf7, 0x4b6d1856, 0x26a36631,
0xeae397b2, 0x3a6efa74, 0xdd5b4332,
                            0x6841e7f7, 0xca7820fb, 0xfb0af54e,
0xd8feb397, 0x454056ac, 0xba489527,
                            0x55533a3a, 0x20838d87, 0xfe6ba9b7,
0xd096954b, 0x55a867bc, 0xa1159a58,
                            0xcca92963, 0x99e1db33, 0xa62a4a56,
0x3f3125f9, 0x5ef47e1c, 0x9029317c,
                            0xfdf8e802, 0x04272f70, 0x80bb155c,
0x05282ce3, 0x95c11548, 0xe4c66d22,
                            0x48c1133f, 0xc70f86dc, 0x07f9c9ee,
0x41041f0f, 0x404779a4, 0x5d886e17,
                            0x325f51eb, 0xd59bc0d1, 0xf2bcc18f,
0x41113564, 0x257b7834, 0x602a9c60,
                            0xdff8e8a3, 0x1f636c1b, 0x0e12b4c2,
0x02e1329e, 0xaf664fd1, 0xcad18115,
                            0x6b2395e0, 0x333e92e1, 0x3b240b62,
0xeebeb922, 0x85b2a20e, 0xe6ba0d99,
                            0xde720c8c, 0x2da2f728, 0xd0127845,
0x95b794fd, 0x647d0862, 0xe7ccf5f0,
                            0x5449a36f, 0x877d48fa, 0xc39dfd27,
0xf33e8d1e, 0x0a476341, 0x992eff74,
                            0x3a6f6eab, 0xf4f8fd37, 0xa812dc60,
0xa1ebddf8, 0x991be14c, 0xdb6e6b0d,
                            0xc67b5510, 0x6d672c37, 0x2765d43b,
0xdcd0e804, 0xf1290dc7, 0xcc00ffa3,
                            0xb5390f92, 0x690fed0b, 0x667b9ffb,
0xcedb7d9c, 0xa091cf0b, 0xd9155ea3,
                            0xbb132f88, 0x515bad24, 0x7b9479bf,
0x763bd6eb, 0x37392eb3, 0xcc115979,
                            0x8026e297, 0xf42e312d, 0x6842ada7,
0xc66a2b3b, 0x12754ccc, 0x782ef11c,
                            0x6a124237, 0xb79251e7, 0x06a1bbe6,
0x4bfb6350, 0x1a6b1018, 0x11caedfa,
                            0x3d25bdd8, 0xe2e1c3c9, 0x44421659,
0x0a121386, 0xd90cec6e, 0xd5abea2a,
                            0x64af674e, 0xda86a85f, 0xbebfe988,
0x64e4c3fe, 0x9dbc8057, 0xf0f7c086,
                            0x60787bf8, 0x6003604d, 0xd1fd8346,
0xf6381fb0, 0x7745ae04, 0xd736fccc,
                            0x83426b33, 0xf01eab71, 0xb0804187,
0x3c005e5f, 0x77a057be, 0xbde8ae24,
                            0x55464299, 0xbf582e61, 0x4e58f48f,
0xf2ddfda2, 0xf474ef38, 0x8789bdc2,
                            0x5366f9c3, 0xc8b38e74, 0xb475f255,
0x46fcd9b9, 0x7aeb2661, 0x8b1ddf84,
```

```
                            0x846a0e79, 0x915f95e2, 0x466e598e,
0x20b45770, 0x8cd55591, 0xc902de4c,
                            0xb90bace1, 0xbb8205d0, 0x11a86248,
0x7574a99e, 0xb77f19b6, 0xe0a9dc09,
                            0x662d09a1, 0xc4324633, 0xe85a1f02,
0x09f0be8c, 0x4a99a025, 0x1d6efe10,
                            0x1ab93d1d, 0x0ba5a4df, 0xa186f20f,
0x2868f169, 0xdcb7da83, 0x573906fe,
                            0xa1e2ce9b, 0x4fcd7f52, 0x50115e01,
0xa70683fa, 0xa002b5c4, 0x0de6d027,
                            0x9af88c27, 0x773f8641, 0xc3604c06,
0x61a806b5, 0xf0177a28, 0xc0f586e0,
                            0x006058aa, 0x30dc7d62, 0x11e69ed7,
0x2338ea63, 0x53c2dd94, 0xc2c21634,
                            0xbbcbee56, 0x90bcb6de, 0xebfc7da1,
0xce591d76, 0x6f05e409, 0x4b7c0188,
                            0x39720a3d, 0x7c927c24, 0x86e3725f,
0x724d9db9, 0x1ac15bb4, 0xd39eb8fc,
                            0xed545578, 0x08fca5b5, 0xd83d7cd3,
0x4dad0fc4, 0x1e50ef5e, 0xb161e6f8,
                            0xa28514d9, 0x6c51133c, 0x6fd5c7e7,
0x56e14ec4, 0x362abfce, 0xddc6c837,
                            0xd79a3234, 0x92638212, 0x670efa8e,
0x406000e0
                    },
                    {
                            0x3a39ce37, 0xd3faf5cf, 0xabc27737,
0x5ac52d1b, 0x5cb0679e, 0x4fa33742,
                            0xd3822740, 0x99bc9bbe, 0xd5118e9d,
0xbf0f7315, 0xd62d1c7e, 0xc700c47b,
                            0xb78c1b6b, 0x21a19045, 0xb26eb1be,
0x6a366eb4, 0x5748ab2f, 0xbc946e79,
                            0xc6a376d2, 0x6549c2c8, 0x530ff8ee,
0x468dde7d, 0xd5730a1d, 0x4cd04dc6,
                            0x2939bbdb, 0xa9ba4650, 0xac9526e8,
0xbe5ee304, 0xa1fad5f0, 0x6a2d519a,
                            0x63ef8ce2, 0x9a86ee22, 0xc089c2b8,
0x43242ef6, 0xa51e03aa, 0x9cf2d0a4,
                            0x83c061ba, 0x9be96a4d, 0x8fe51550,
0xba645bd6, 0x2826a2f9, 0xa73a3ae1,
                            0x4ba99586, 0xef5562e9, 0xc72fefd3,
0xf752f7da, 0x3f046f69, 0x77fa0a59,
                            0x80e4a915, 0x87b08601, 0x9b09e6ad,
0x3b3ee593, 0xe990fd5a, 0x9e34d797,
                            0x2cf0b7d9, 0x022b8b51, 0x96d5ac3a,
0x017da67d, 0xd1cf3ed6, 0x7c7d2d28,
                            0x1f9f25cf, 0xadf2b89b, 0x5ad6b472,
0x5a88f54c, 0xe029ac71, 0xe019a5e6,
                            0x47b0acfd, 0xed93fa9b, 0xe8d3c48d,
0x283b57cc, 0xf8d56629, 0x79132e28,
                            0x785f0191, 0xed756055, 0xf7960e44,
0xe3d35e8c, 0x15056dd4, 0x88f46dba,
                            0x03a16125, 0x0564f0bd, 0xc3eb9e15,
0x3c9057a2, 0x97271aec, 0xa93a072a,
```

```
                            0x1b3f6d9b, 0x1e6321f5, 0xf59c66fb,
0x26dcf319, 0x7533d928, 0xb155fdf5,
                            0x03563482, 0x8aba3cbb, 0x28517711,
0xc20ad9f8, 0xabcc5167, 0xccad925f,
                            0x4de81751, 0x3830dc8e, 0x379d5862,
0x9320f991, 0xea7a90c2, 0xfb3e7bce,
                            0x5121ce64, 0x774fbe32, 0xa8b6e37e,
0xc3293d46, 0x48de5369, 0x6413e680,
                            0xa2ae0810, 0xdd6db224, 0x69852dfd,
0x09072166, 0xb39a460a, 0x6445c0dd,
                            0x586cdecf, 0x1c20c8ae, 0x5bbef7dd,
0x1b588d40, 0xccd2017f, 0x6bb4e3bb,
                            0xdda26a7e, 0x3a59ff45, 0x3e350a44,
0xbcb4cdd5, 0x72eacea8, 0xfa6484bb,
                            0x8d6612ae, 0xbf3c6f47, 0xd29be463,
0x542f5d9e, 0xaec2771b, 0xf64e6370,
                            0x740e0d8d, 0xe75b1357, 0xf8721671,
0xaf537d5d, 0x4040cb08, 0x4eb4e2cc,
                            0x34d2466a, 0x0115af84, 0xe1b00428,
0x95983a1d, 0x06b89fb4, 0xce6ea048,
                            0x6f3f3b82, 0x3520ab82, 0x011a1d4b,
0x277227f8, 0x611560b1, 0xe7933fdc,
                            0xbb3a792b, 0x344525bd, 0xa08839e1,
0x51ce794b, 0x2f32c9b7, 0xa01fbac9,
                            0xe01cc87e, 0xbcc7d1f6, 0xcf0111c3,
0xa1e8aac7, 0x1a908749, 0xd44fbd9a,
                            0xd0dadecb, 0xd50ada38, 0x0339c32a,
0xc6913667, 0x8df9317c, 0xe0b12b4f,
                            0xf79e59b7, 0x43f5bb3a, 0xf2d519ff,
0x27d9459c, 0xbf97222c, 0x15e6fc2a,
                            0x0f91fc71, 0x9b941525, 0xfae59361,
0xceb69ceb, 0xc2a86459, 0x12baa8d1,
                            0xb6c1075e, 0xe3056a0c, 0x10d25065,
0xcb03a442, 0xe0ec6e0e, 0x1698db3b,
                            0x4c98a0be, 0x3278e964, 0x9f1f9532,
0xe0d392df, 0xd3a0342b, 0x8971f21e,
                            0x1b0a7441, 0x4ba3348c, 0xc5be7120,
0xc37632d8, 0xdf359f8d, 0x9b992f2e,
                            0xe60b6f47, 0x0fe3f11d, 0xe54cda54,
0x1edad891, 0xce6279cf, 0xcd3e7e6f,
                            0x1618b166, 0xfd2c1d05, 0x848fd2c5,
0xf6fb2299, 0xf523f357, 0xa6327623,
                            0x93a83531, 0x56cccd02, 0xacf08162,
0x5a75ebb5, 0x6e163697, 0x88d273cc,
                            0xde966292, 0x81b949d0, 0x4c50901b,
0x71c65614, 0xe6c6c7bd, 0x327a140a,
                            0x45e1d006, 0xc3f27b9a, 0xc9aa53fd,
0x62a80f00, 0xbb25bfe2, 0x35bdd2f6,
                            0x71126905, 0xb2040222, 0xb6cbcf7c,
0xcd769c2b, 0x53113ec0, 0x1640e3d3,
                            0x38abbd60, 0x2547adf0, 0xba38209c,
0xf746ce76, 0x77afa1c5, 0x20756060,
                            0x85cbfe4e, 0x8ae88dd8, 0x7aaaf9b0,
0x4cf9aa7e, 0x1948c25c, 0x02fb8a8c,
```

```
                            0x01c36ae4, 0xd6ebe1f9, 0x90d4f869,
0xa65cdea0, 0x3f09252d, 0xc208e69f,
                            0xb74e6132, 0xce77e25b, 0x578fdfe3,
0x3ac372e6
                    }
              };

        uint[] P;
        uint[,] S;
   /// <summary>
        /// Constructs and initializes a blowfish instance with the
supplied key.
        /// </summary>
        /// <param name="key">The key to cipher with.</param>
        public Blowfish(byte[] key)
        {
            short i;
            short j;
            short k;
            uint data;
            uint datal;
            uint datar;

            P = _P.Clone() as uint[];
            S = _S.Clone() as uint[,];

            j = 0;
            for (i = 0; i < N + 2; ++i)
            {
                data = 0x00000000;
                for (k = 0; k < 4; ++k)
                {
                    data = (data << 8) | key[j];
                    j++;
                    if (j >= key.Length)
                    {
                        j = 0;
                    }
                }
                P[i] = P[i] ^ data;
            }

            datal = 0x00000000;
            datar = 0x00000000;

            for (i = 0; i < N + 2; i += 2)
            {
                Encipher(ref datal, ref datar);
                P[i] = datal;
                P[i + 1] = datar;
            }

            for (i = 0; i < 4; ++i)
            {
```

```
                for (j = 0; j < 256; j += 2)
                {
                    Encipher(ref datal, ref datar);

                    S[i, j] = datal;
                    S[i, j + 1] = datar;
                }
            }
        }

        /// <summary>
        /// Constructs and initializes a blowfish instance with the
supplied key.
        /// </summary>
        /// <param name="key">The key to cipher with.</param>
        public            Blowfish(string            key)            :
this(Encoding.Unicode.GetBytes(key))
        {

        }

        /// <summary>
        ///
        /// </summary>
        /// <param name="x"></param>
        /// <returns></returns>
        private uint F(uint x)
        {
            ushort a;
            ushort b;
            ushort c;
            ushort d;
            uint y;

            d = (ushort)(x & 0x00FF);
            x >>= 8;
            c = (ushort)(x & 0x00FF);
            x >>= 8;
            b = (ushort)(x & 0x00FF);
            x >>= 8;
            a = (ushort)(x & 0x00FF);
            //y = ((S[0][a] + S[1][b]) ^ S[2][c]) + S[3][d];
            y = S[0, a] + S[1, b];
            y = y ^ S[2, c];
            y = y + S[3, d];

            return y;
        }

        /// <summary>
        /// Encrypts a byte array in place.
        /// </summary>
        /// <param name="data">The array to encrypt.</param>
        /// <param name="length">The amount to encrypt.</param>
```

```
        public void Encipher(byte[] data, int length)
        {
            uint xl, xr;
            if ((length % 8) != 0)
                throw new Exception("Invalid Length");
            for (int i = 0; i < length; i += 8)
            {
                // Encode the data in 8 byte blocks.
                xl = (uint)((data[i] << 24) | (data[i + 1] << 16) |
(data[i + 2] << 8) | data[i + 3]);
                xr = (uint)((data[i + 4] << 24) | (data[i + 5] << 16) |
(data[i + 6] << 8) | data[i + 7]);
                Encipher(ref xl, ref xr);
                // Now Replace the data.
                data[i] = (byte)(xl >> 24);
                data[i + 1] = (byte)(xl >> 16);
                data[i + 2] = (byte)(xl >> 8);
                data[i + 3] = (byte)(xl);
                data[i + 4] = (byte)(xr >> 24);
                data[i + 5] = (byte)(xr >> 16);
                data[i + 6] = (byte)(xr >> 8);
                data[i + 7] = (byte)(xr);
            }
        }

        /// <summary>
        /// Encrypts 8 bytes of data (1 block)
        /// </summary>
        /// <param name="xl">The left part of the 8 bytes.</param>
        /// <param name="xr">The right part of the 8 bytes.</param>
        private void Encipher(ref uint xl, ref uint xr)
        {
            uint Xl;
            uint Xr;
            uint temp;
            short i;

            Xl = xl;
            Xr = xr;

            for (i = 0; i < N; ++i)
            {
                Xl = Xl ^ P[i];
                Xr = F(Xl) ^ Xr;

                temp = Xl;
                Xl = Xr;
                Xr = temp;
            }

            temp = Xl;
            Xl = Xr;
            Xr = temp;
```

```
            Xr = Xr ^ P[N];
            Xl = Xl ^ P[N + 1];

            xl = Xl;
            xr = Xr;
        }

        /// <summary>
        /// Encrypts string
        /// </summary>
        /// <param name="data">The string to encrypt</param>
        /// <returns>Encrypted string</returns>
        public String Encipher(String data)
        {
            byte[] b = Encoding.Unicode.GetBytes(data);
            Encipher(b, b.Length);

            return Convert.ToBase64String(b);
        }

        /// <summary>
        /// Decrypts a byte array in place.
        /// </summary>
        /// <param name="data">The array to decrypt.</param>
        /// <param name="length">The amount to decrypt.</param>
        public void Decipher(byte[] data, int length)
        {
            uint xl, xr;
            if ((length % 8) != 0)
                throw new Exception("Invalid Length");
            for (int i = 0; i < length; i += 8)
            {
                // Encode the data in 8 byte blocks.
                xl = (uint)((data[i] << 24) | (data[i + 1] << 16) |
(data[i + 2] << 8) | data[i + 3]);
                xr = (uint)((data[i + 4] << 24) | (data[i + 5] << 16) |
(data[i + 6] << 8) | data[i + 7]);
                Decipher(ref xl, ref xr);
                // Now Replace the data.
                data[i] = (byte)(xl >> 24);
                data[i + 1] = (byte)(xl >> 16);
                data[i + 2] = (byte)(xl >> 8);
                data[i + 3] = (byte)(xl);
                data[i + 4] = (byte)(xr >> 24);
                data[i + 5] = (byte)(xr >> 16);
                data[i + 6] = (byte)(xr >> 8);
                data[i + 7] = (byte)(xr);
            }
        }

        /// <summary>
        /// Decrypts 8 bytes of data (1 block)
        /// </summary>
        /// <param name="xl">The left part of the 8 bytes.</param>
```

```
    /// <param name="xr">The right part of the 8 bytes.</param>
    private void Decipher(ref uint xl, ref uint xr)
    {
        uint Xl;
        uint Xr;
        uint temp;
        short i;

        Xl = xl;
        Xr = xr;

        for (i = N + 1; i > 1; --i)
        {
            Xl = Xl ^ P[i];
            Xr = F(Xl) ^ Xr;

            /* Exchange Xl and Xr */
            temp = Xl;
            Xl = Xr;
            Xr = temp;
        }

        /* Exchange Xl and Xr */
        temp = Xl;
        Xl = Xr;
        Xr = temp;

        Xr = Xr ^ P[1];
        Xl = Xl ^ P[0];

        xl = Xl;
        xr = Xr;
    }

    /// <summary>
    /// Decrypt string
    /// </summary>
    /// <param name="data">The String to decrypt</param>
    /// <returns>Decrypted string</returns>
    public String Decipher(String data)
    {
        byte[] b = Convert.FromBase64String(data);
        Decipher(b, b.Length);

        return Encoding.Unicode.GetString(b);
    }
}

public class BlowfishStream : Stream
{
    class CBState : IAsyncResult
    {
        internal AsyncCallback callback;
        internal object state;
```

```
            internal byte[] buffer;
            internal IAsyncResult result;
            internal  CBState(AsyncCallback  callback,  object  state,
byte[] buffer)
            {
                this.callback = callback;
                this.state = state;
                this.buffer = buffer;
            }
            #region IAsyncResult Members

            public object AsyncState
            {
                get
                {
                    return state;
                }
            }

            public bool CompletedSynchronously
            {
                get
                {
                    return result.CompletedSynchronously;
                }
            }

            public System.Threading.WaitHandle AsyncWaitHandle
            {
                get
                {
                    return result.AsyncWaitHandle;
                }
            }

            public bool IsCompleted
            {
                get
                {
                    return result.IsCompleted;
                }
            }

            #endregion
        }

        public enum Target
        {
            Encrypted,
            Normal
        };

        Stream stream;
        Blowfish bf;
```

```
Target target;

BlowfishStream(Stream stream, Blowfish bf, Target target)
{
    this.stream = stream;
    this.bf = bf;
    this.target = target;
}

/// <summary>
/// Returns true if the stream support reads.
/// </summary>
public override bool CanRead
{
    get { return stream.CanRead; }
}

/// <summary>
/// Returns true is the stream supports seeks.
/// </summary>
public override bool CanSeek
{
    get { return stream.CanSeek; }
}

/// <summary>
/// Returns true if the stream supports writes.
/// </summary>
public override bool CanWrite
{
    get { return stream.CanWrite; }
}

/// <summary>
/// Returns the length of the stream.
/// </summary>
public override long Length
{
    get { return stream.Length; }
}

/// <summary>
/// Gets or Sets the posistion of the stream.
/// </summary>
public override long Position
{
    get { return stream.Position; }
    set { stream.Position = value; }
}

/// <summary>
/// Flushes the stream.
/// </summary>
public override void Flush()
```

```
        {
            stream.Flush();
        }

        /// <summary>
        /// Read data from the stream and encrypt it.
        /// </summary>
        /// <param name="buffer">The buffer to read into.</param>
        /// <param name="offset">The offset in the buffer to begin
storing data.</param>
        /// <param name="count">The number of bytes to read.</param>
        /// <returns></returns>
        public override int Read(byte[] buffer, int offset, int count)
        {
            int bytesRead = stream.Read(buffer, offset, count);
            if (target == Target.Normal)
                bf.Encipher(buffer, bytesRead);
            else
                bf.Decipher(buffer, bytesRead);
            return bytesRead;
        }

        /// <summary>
        /// Write data to the stream after decrypting it.
        /// </summary>
        /// <param name="buffer">The buffer containing the data to
write.</param>
        /// <param name="offset">The offset in the buffer where the data
begins.</param>
        /// <param name="count">The number of bytes to write.</param>
        public override void Write(byte[] buffer, int offset, int count)
        {
            if (target == Target.Normal)
                bf.Decipher(buffer, count);
            else
                bf.Encipher(buffer, count);
            stream.Write(buffer, offset, count);
        }

        /// <summary>
        ///
        /// </summary>
        /// <param name="buffer"></param>
        /// <param name="offset"></param>
        /// <param name="count"></param>
        /// <param name="callback"></param>
        /// <param name="state"></param>
        /// <returns></returns>
        public override IAsyncResult BeginRead(byte[] buffer, int
offset, int count, AsyncCallback callback, object state)
        {
            CBState cbs = new CBState(callback, state, buffer);
            cbs.result = base.BeginRead(buffer, offset, count, new
AsyncCallback(ReadComplete), cbs);
```

```
            return cbs;
        }

        /// <summary>
        ///
        /// </summary>
        /// <param name="asyncResult"></param>
        /// <returns></returns>
        public override int EndRead(IAsyncResult asyncResult)
        {
            CBState cbs = (CBState)asyncResult.AsyncState;
            int bytesRead = base.EndRead(cbs.result);
            if (target == Target.Normal)
                bf.Encipher(cbs.buffer, bytesRead);
            else
                bf.Decipher(cbs.buffer, bytesRead);
            return bytesRead;
        }


        /// <summary>
        /// The Read has completed.
        /// </summary>
        /// <param name="result">The result of the async write.</param>
        private void ReadComplete(IAsyncResult result)
        {
            CBState cbs = (CBState)result.AsyncState;
            cbs.callback(cbs);
        }


        /// <summary>
        ///
        /// </summary>
        /// <param name="buffer"></param>
        /// <param name="offset"></param>
        /// <param name="count"></param>
        /// <param name="callback"></param>
        /// <param name="state"></param>
        /// <returns></returns>
        public override IAsyncResult BeginWrite(byte[] buffer, int
offset, int count, AsyncCallback callback, object state)
        {
            if (target == Target.Normal)
                bf.Decipher(buffer, count);
            else
                bf.Encipher(buffer, count);
            return base.BeginWrite(buffer, offset, count, callback,
state);
        }

        /// <summary>
        /// Move the current stream posistion to the specified location.
        /// </summary>
```

```
        /// <param name="offset">The offset from the origin to
    seek.</param>
        /// <param name="origin">The origin to seek from.</param>
        /// <returns>The new position.</returns>
        public override long Seek(long offset, SeekOrigin origin)
        {
            return stream.Seek(offset, origin);
        }

        /// <summary>
        /// Set the stream length.
        /// </summary>
        /// <param name="value">The length to set.</param>
        public override void SetLength(long value)
        {
            stream.SetLength(value);
        }
    }
}
```

2. Modified Blowfish

The code of modified F-function in c#

```
private uint F(uint x)
    {
        ushort a;
        ushort b;

        uint y;

        b = (ushort)(x & 0x00FF);
        x >>= 16;
        a = (ushort)(x & 0x00FF);

        y = S[0, a] ^ S[1, b];

        return y;

    }
```