Implementation of Tiny Encryption Algorithm For Securing System File

A

DISSERTATION

Submitted as a course in partial fulfillment of the

requirements for the award of the degree of

MASTER OF ENGINEERING

In

Computer Technology Application

Submitted By
Raman Kumar
Roll No: 13/CTA/09
University Roll No: 12210

Under the Esteemed Guidance

Of

Mr. Manoj Sethi
(Head of Computer Center)
Delhi College of engineering

Department of Computer Engineering
DELHI COLLEGE OF ENGINEERING
BAWANA ROAD, NEW DELHI-110042
2007-2009 (University of Delhi)

CERTIFICATE

It is certified that the project entitled "**Implementation of Tiny Encryption Algorithm for Securing System File**" being submitted by **Mr. Raman Kumar** M.E. in Computer Technology Application, **Delhi College of Engineering** is work carried out by his under my guidance and supervision in partial fulfillment for the award of the degree of Master of Engineering in Computer Technology and Applications from Delhi College of Engineering, Delhi.

Lastly, I thank Almighty GOD for his countless blessings.

Mr. Manoj Sethi
Head of Computer Center
Delhi College of Engineering
Bawana Road, New Delhi

ACKNOWLEDGEMENT

It gives me immense pleasure in expressing my deep sense of gratitude, indebtness and thankfulness to **Mr.Manoj Sethi** (Head of Computer Centre) for his invaluable guidance, continual encouragement and support at every stage of this work. I am also thankful to Dr. Daya Gupta Head of Computer Department for the encouragement given throughout the execution of this project.

I would also like to express my sincere thanks to my teachers' viz. Dr S. K. Saxena, Mrs. Rajni Jindal. Mr.Shailender Verma for their support and encouragement.

I would like to express my heartiest felt regards to Faculty of Computer Engineering Department for providing facility and for their co-operation and patience.

I am thankful to my friends and classmates for their unconditional support and motivation during this project.

Finally I acknowledge my deep gratitude to my loving family, who always gave moral support and continuously encourage my academic endeavor, who always comfort and console, never complain or interfere, ask nothing and endure all.

Raman Kumar
University Roll No: 12210
College Roll No: 13/CTA/ 07
Delhi College of Engineering
Bawana Road, New Delhi

ABSTRACT

The file securing system requires a Tiny Encryption Algorithm (TEA). It basically consist of using a combination of XOR, ADD and SHIFT operations on file blocks to encrypt is using key bits and some junk bits. This algorithm is highly efficient and can satisfy need for a user-friendly encryption system that can be used at homes and offices.

The Tiny Encryption Algorithm (TEA) is a block cipher encryption algorithm that is very simple to implement, has fast execution time, and takes minimal storage space.

The Tiny Encryption Algorithm (TEA) is a cryptographic algorithm designed to minimize memory footprint and maximize speed. It is a Feistel type cipher that uses operations from mixed (orthogonal) algebraic groups. This thesis presents the cryptanalysis of the Tiny Encryption Algorithm. In this thesis proposed to inspect the most common methods in the cryptanalysis of a block cipher algorithm. TEA seems to be highly resistant to differential cryptanalysis, and achieves complete diffusion (where a one bit difference in the plaintext will cause approximately 32 bit differences in the cipher text) after only six rounds. Time performance on a modern desktop computer or workstation is very impressive.

The objective is not to make it difficult to read the data as cryptography does it is hide the existence of the data in the first place possible to protect the courier.

Tiny Encryption algorithm has been suggested by researcher. It is proposed to carry forward the work for attempt to implement Tiny Encryption Algorithm for confidentiality, authenticity of system files.

File Encryption System (using Tiny Encryption Algorithm) proposed to consist of following functions:-

- User friendly encryption of text files.
- Encryption of images.
- Decryption of Encrypted files**.**

Table of Contents

CERTIFICATE ……………………………………………………………………i

ACKNOWLEDGEMENT…………………………………………………………ii

ABSTRACT ………………………………………………………………….iii

CHAPTER No **Page No.**

CHAPTER 1: - Introduction ………………………………………………….…..1

1.1 Encryption and Decryption ………… ……………………….…….1

1.2 History of Cryptography ……………………………………… .2

1.3 Classical Cryptography …………………………………….………3

1.4 Medieval Cryptography…………………………………….....4

1.5 Modern Cryptography ………………………………………. ….5

1.6 Application of Cryptography……………………………………6

1.7 Need of Encryption……………………………………………8

CHAPTER 2: - Literature Review ……………………………………..….….. 9

2.1 Symmetrical Cryptography Algorithms……………………..………9

2.1.1 Introduction…………………………………………………..9

2.1.2 TEA, AES, DES, Symmetrical Algorithms ………………... …….11

2.1.3 Symmetric Encryption vulnerabilities………………… ………..13

CHAPTER 3: - Proposed Approach……………………………………..…….…14

3.1 Background and Motivation ……………………………… …………14

3.2 TEA Encryption and Decryption algorithm…………………………… 16

3.2.1 Encryption Routine……………………………………………...16

3.2.2 Decryption Routine……………………………………………20

3.2.3 Basics of the routine…………………………………………..…..…22

3.2.4 A Security Algorithm TEA……………………………………….23

3.3 Proposed Work………………………………………………………..24

3.4 Implementation of TEA…………………………………………… …24

3.4.1 Requirement to Implement New algorithm ……………………….24

CHAPTER: - 4 Conclusion and Future Works……………………………………..28

References………………………………………………………………………....30

Appendix-1 Data Flow Diagram of File Encryption System ………………….…32

Appendix-2 Source Code…………………………………………………….…...34

Appendix -3 Result and Snapshot …………………………………………….…53

CHAPTER :-1
Intoduction

1.1 Encryption and Decryption

A more flexible security approach - a combination of message encryption (sealing) and message signing (digital signatures).

The key to Encryption

A Key is a set of characters of varying length that is used with the cipher encryption formula) to generate encrypted data. There are two relevant types of keys:

1. Secret Key: - It is much like a secret password. The same password that is used to encrypt the file is also used to decrypt it. Secret keys are also called symmetric keys, because there is only one key for encryption and decryption.

The advantage of a secret key cipher is:

It is generally very fast: a computer can encrypt a large amount of data in a very short time using a secret key cipher.

2. Public/Private Key pair: - It is a special type of key relationship. This arrangement consists of two very large keys that are mathematically related to one another, but the mathematical relationship is very difficult to calculate. Only one key is given to anyone who wants (public key), and the other key are accessed by the user (private key).

The advantage to public/private key encryption is:

1. Secure: The keys are much more secured since one alone has a control access to the private key.
2. Hard to decrypt: Also, since the key size is so large, the encrypted data is very hard to decrypt.

The disadvantage to public/private key encryption is:

1. Time consuming: Since the key size is so large, encryption and decryption is very time consuming and CPU-intensive. Hashing is a mathematical function that is applied to a string of characters of any length, or to an entire file. The hashing function reduces any length of characters to a fixed length. Hashing is also called a message digest, because the hashed value uniquely represents the original data. Hashing functions are commonly used in message signing (digital signatures) to create a unique "signature" of the message in question. There are a number of algorithms that are used to create a message digest, including RSAs MD2, MD4 and MD5 which all create a 128-bit hash.

1.2 History of Cryptography

The history of cryptography begins thousands of years ago. Until recent decades, it has been the story of what might be called classic cryptography — that is, of methods of encryption that use pen and paper, or perhaps simple mechanical aids. In the early 20th century, the invention of complex mechanical and electromechanical machines, such as the Enigma rotor machine, provided more sophisticated and efficient means of encryption; and the subsequent introduction of electronics and computing has allowed elaborate schemes of still greater complexity, most of which are entirely unsuited to pen and paper.[20]

The development of cryptography has been paralleled by the development of cryptanalysis — of the "breaking" of codes and ciphers. The discovery and application, early on, of frequency analysis to the reading of encrypted communications has on occasion altered the course of history. Thus the Zimmermann Telegram triggered the United States' entry into World War I; and Allied reading of Nazi Germany's ciphers shortened World War II, in some evaluations by as much as two years.

Until the 1970s, secure cryptography was largely the preserve of governments. Two events have since brought it squarely into the public domain: the creation of a public encryption standard (DES); and the invention of public-key cryptography[21].

1.3 Classical Cryptography

The earliest known use of cryptography is found in non-standard hieroglyphs carved into monuments from Egypt's Old Kingdom (c 4500+ years ago). These are not thought to be serious attempts at secret communications, however, but rather to have been attempts at mystery, intrigue, or even amusement for literate onlookers. These are examples of still other uses of cryptography, or of something that looks (impressively if misleadingly) like it. Some clay tablets from Mesopotamia, somewhat later are clearly meant to protect information -- they encrypt recipes, presumably commercially valuable. Later still, Hebrew scholars made use of simple monoalphabetic substitution ciphers (such as the Atbash cipher) beginning perhaps around 500 to 600 BCE.[22]

Cryptography has a long tradition in religious writing likely to offend the dominant culture or political authorities. Perhaps the most famous is the 'Number of the Beast' from the Book of Revelations in the Christian New Testament. '666' might be a cryptographic (i.e., encrypted) way of concealing a dangerous reference; many scholars believe it's a concealed reference to the Roman Empire, or more likely to the Emperor Nero himself, (and so to Roman persecution policies) that would have been understood by the initiated (who 'had the key to understanding'), and yet be safe or at least somewhat deniable (and so 'less' dangerous) if it came to the attention of the authorities. At least for orthodox Christian writing, most of the need for such concealment ended with Constantine's conversion and the adoption of Christianity as the official religion of the Empire.[21]

1.4 Medieval Cryptography

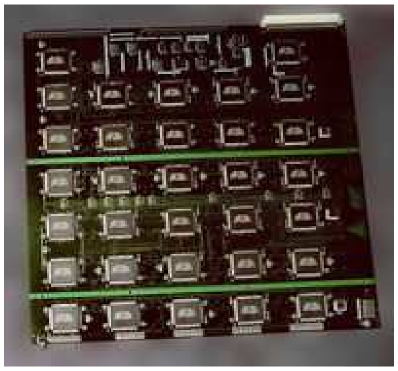
Although Alberti is usually considered the father of polyalphabetic cipher, Prof. Al-Kadi's 1990 paper (ref- 3), reviewing Arabic contributions to cryptography reported knowledge of polyalphabetic ciphers 500 years before Alberti, based on a recently discovered manuscript). It appears that Abu Yusuf Yaqub ibn Is-haq ibn as Sabbah ibn 'omran ibn Ismail Al- Kindi, who wrote a book on cryptography called "Risalah fi Istikhraj al-Mu'amma" (Manuscript for the Deciphering Cryptographic Messages), circa 750 CE), may have described cryptanalysis techniques (including some for polyalphabetic ciphers), cipher classification, Arabic Phonetics and Syntax, and, most importantly, described the use of several statistical techniques for cryptanalysis. [This book seems to be the first post-classical era reference by about 300 years.] It also contains probability and statistical work some 800 years before Pascal and Fermat. Cryptography became (secretly) still more important as a consequence of political competition and religious revolution. For instance, in Europe during and after the Renaissance, citizens of the various Italian states -- the Papal States and the Roman Catholic Church included -- were responsible for rapid proliferation of cryptographic techniques, few of which reflect understanding (or even knowledge) of Alberti's polyalphabetic advance. 'Advanced ciphers', even after Alberti, weren't as advanced as their inventors / developers / users claimed (and probably even themselves believed). They were regularly broken. This over-optimism may be inherent in cryptography for it was then, and remains today, fundamentally difficult to really know how vulnerable your system actually is. In the absence of knowledge, guesses and hopes, as may be expected, are common.

Cryptography, cryptanalysis, and secret agent/courier betrayal featured in the Babington plot during the reign of Queen Elizabeth I which led to the execution of Mary, Queen of Scots. An encrypted message from the time of the Man in the Iron Mask (decrypted just prior to 1900 by Étienne Bazeries) has shed some, regrettably non-definitive, light on the identity of that real, if legendary and unfortunate, prisoner. Cryptography, and its misuse, were involved in the plotting which led to the execution of Mata Hari and in the conniving which led to the travesty of Dreyfus' conviction and imprisonment, both in the early 20th century. Fortunately, cryptographers were also involved in exposing the machinations which had led to Dreyfus' problems; Mata Hari, in contrast, was shot.[19] Outside of Europe, after the end of the Muslim Golden Age at the hand of the Mongols, cryptography remained comparatively undeveloped. Cryptography in Japan seems not to have been used until about 1510, and advanced techniques were not known until after the opening of the country to the West beginning in the 1860s.[22]

1.5 Modern Cryptography

The era of modern cryptography really begins with Claude Shannon, arguably the father of mathematical cryptography, with the work he did during WWII on communications security. In 1949 he published the paper Communication Theory of Secrecy Systems in the Bell System Technical Journal and a little later the book, Mathematical Theory of Communication, with Warren Weaver. both included results from his WWII work. These, in addition to his other works on information and communication theory established a solid theoretical basis for cryptography and for cryptanalysis. And with that, cryptography more or less disappeared into secret government communications organizations such as the NSA, GCHQ, and equivalents elsewhere. Very little work was again made public until the mid '70s, when everything changed.

Modern Cryptanalysis



Modern cryptanalysts sometimes harness large numbers of circuits. This board is part of the EFF DES cracker, which contained over 1800customchips and could brute force a DES key in a matter of days.

While modern ciphers like AES are widely considered unbreakable, poor designs are still sometimes adopted and there have been important cryptanalytic breaks of deployed crypto systems in recent years. Notable examples of broken crypto designs include DES, the first Wi-Fi encryption scheme WEP, the Content Scrambling System used for encrypting and controlling DVD use, and the A5/1 and A5/2 ciphers used in GSM cell phones. Thus far, not one of the mathematical ideas underlying public key cryptography has been proven to be 'unbreakable' and so some future advance might render systems relying on them insecure. While few informed observers foresee such a breakthrough, the key size recommended for security keeps increasing as increased computing power required for breaking codes becomes cheaper and more available. [22]

1.6 Application of Cryptography

Some businesses began using strong cryptography about twenty years ago. They seemed to do it in the half-instinctive way they used office safes and armored-car services. Banks used DES (the US Data Encryption Standard) to secure the electronic transfers between branches, and later in ATMs. And any business with a computer had some kind of password system, either to control access to the computer or to certain disk files. It was just done. No one made much fuss about it. Little by little, things changed. Very strong cryptography left the shadows of national security organizations and started to look like an essential business tool -- not least for exercising a 'duty of care' for information in stored electronic files or sent over electronic networks.[19]

Business use of encryption will keep growing. There are three main reasons:

1. Computers have changed greatly.

Twenty-five years ago most computers were centralized, in locked rooms and were looked after by people with arcane vocabularies. An electronic link to the outside was unusual. And if there was a link, it was along a dedicated line. Security threats in those days were mostly from insiders: people abusing their accounts, theft of data and sometimes vandalism. These threats were managed by keeping the computers behind locked doors and accounting scrupulously for resources. Today computers are here, there and everywhere, including people's private offices. Most computers are now connected into networks. So central management isn't feasible and security is harder to manage. Much harder.

3. **Messages and electronic files now move along insecure networks, not just along dedicated lines.**

There is no security on the Internet. And even an internal LAN can be broken into if there's just one insecure dial-in modem.

4. **Faxes have proved hard to manage for sending confidential material.**

It is difficult to maintain a 'need to know' system when anyone walking by a fax machine can glance at what comes in. Also, faxes are sometimes sent to the wrong number. And fax interception is now technically simple -- even broadband fax interception from satellite or microwave links. Some fax systems are now sold that encrypt the transmission, but they can leave a manager hovering near the fax machine and waiting for an incoming call -- because the message still comes out in plain view. A smarter system is proving to be point-to-point encryption for email.

1.7 Need of Encryption

As you've learned by now, your transmissions can have only so much physical security. It is reasonable to assume that at some point someone may intercept your transmissions. Whether you expect an interception or whether you just generally suspect that interceptions may occur, you should transmit your information in a format that is useless to any interceptors. At the simplest level, this means when transmitting a message to someone, you use a coded message or slang (nicknames) that no one else understands. When Ulysses S. Grant captured Vicksburg during the Civil War, he sent a coded but predetermined message to Abraham Lincoln that read "The father of waters flows unsexed to the sea," meaning that the Union now owned the whole Mississippi river. Perhaps a good plan at the time, but still, Grant and Lincoln (or their advisers/confidantes) had to communicate a predetermined message and the message's meaning. A more recent example of a coded message might involve the use of nicknames. For instance, you and your sister give nicknames to family members whom you discuss unfavorably. Should a malicious family member decide to intercept a transmission, you would hope he wouldn't understand which family members you and your sister refer to in your messages. The obvious drawback of this coded message, like the Grant-Lincoln message, is that you and the recipient must establish a system of code before you begin transmitting messages. [4]

A better system is one that allows you to send any message, even one you had not anticipated, to anyone without fear of interception. This is why an encryption system is so valuable; it allows any message to be transmitted that will be useless to anyone who intercepts it. [23]

Chapter:-2 Literature Review

A Detailed literature survey shows that there has been a lot of work done in the field of data security system, especially microcontroller in assembly language and hardware design System such as parallel, sequential and digit-serial and there is a great need to improve the Tiny algorithm for the independent platform.[2]

Earlier research's proposed TEA algorithm implement in C language and hardware encryption core such as radio frequency identification (RFID) usually employ public – key algorithm and Assembly language for microcontrollers. The concept of Tiny Encryption Algorithm was given by P. Israsena [1]. The TEA is updated and modified various researchers so that it can be used in different applications.

Research Study

The Tiny Encryption Algorithm (TEA) is a cryptographic algorithm designed by Wheeler and Needham (1994). It is designed to minimize memory footprint and maximize speed.[1] This research presents the cryptanalysis of the Tiny Encryption Algorithm based on the differential cryptanalysis proposed by Biham and Shamir (1992) and related-key cryptanalysis proposed by Kelsey, Schneier, and Wagner (1997) [22].

2.1 Symmetric Cryptography Algorithms

2.2.1 Introduction

Symmetric ***or s***ecret key, cryptography has been in use for thousands of years and includes any form where the same key is used both to encrypt and to decrypt the text involved. One of the simplest forms is sometimes known as the Caesar cipher -- reputedly used by Julius Caesar to conceal messages -- in which the process is simply one of shifting the alphabet so many places in one direction or another. The example given in part 1 of HAL/IBM is in exactly this form with the key being the instruction to shift one letter forwards to decrypt. In this trivial example, the decryption key is a mirror image rather than a replica of the encryption key, but that doesn't vitiate the classification as a symmetric mechanism. [1]

A variation on this simple scheme involves using an arbitrarily ordered alphabet of the same length as the one used for the plain text message. In this case the key might be a long sequence of numbers such as 5, 19, 1, 2, 11 ...[13] indicating that A would map to E, B to S, C to A, D to B, E to K and so on -- or it might be one of a number of more or less ingenious schemes involving letters taken from, say, sentences of particular novels. Such systems are ludicrously weak, of course, and modern systems use sophisticated algorithms based on mathematical problems that are difficult to solve and so tend to be very strong. [3]

Unlike the situation in asymmetric cryptography where there is a public element to the process and where the private key is almost never shared, symmetric cryptography normally requires the key to be shared and simultaneously kept secret within a restricted group. It's simply not possible for a person who views the encrypted data with a symmetric cipher to be able to do so without having access to the key used to encrypt it in the first place. If such a secret key falls into the wrong hands, then the security of the data encrypted using that key is immediately and completely compromised. [4]Hence, what all systems in this group of secret key methods share is the problem of key management, something discussed in more detail in the feature on practical implications (to follow shortly in the series). [22]

Reference is often made to keys of particular bit lengths, such as 56-bit or 128-bit. These lengths are those for symmetric key ciphers, while key lengths for at least the private element of asymmetric ones are considerably longer. Further, there is no correlation between the key lengths in the two groups except incidentally through the perceived level of security which a given key length might offer using a given system. However, Phil Zimmermann, originator of the extremely efficient and important software package known as Pretty Good Privacy (PGP), suggests than an 80-bit symmetric key might approximately equate in security terms at the present moment to a 1024-bit asymmetric key; [5] to gain the security offered by a 128-bit symmetric key, one might need to use a 3000-bit asymmetric key. Others will certainly take issue with some of those comparisons as well as, no doubt, with the attempt even to make them.

Within any particular group, however, the length of the key used is generally a significant element in determining security. Further, key length is not linear but doubles with each additional bit. Two to the power two is four; to the power three is eight, to the power four sixteen, and so on. Giga Group offers a homespun analogy suggesting that if a teaspoon were sufficient to hold all possible 40-bit key combinations, it would take a swimming pool to hold all 56-bit key combinations, while the volume to hold all possible 128-bit key combinations would be roughly equivalent to that of the earth. [8] A 128-bit value, rendered in decimal, is approximately 340 followed by 36 zeros.

2.1.2 TEA, AES/Rijndael (Advanced Encryption Standard) , DES (Digital Encryption Standard), Symmetric Algorithms

Tiny Encryption Algorithm

Wheeler et al. (1994) at the computer laboratory of Cambridge University developed the TEA encode routine. Figure 2.1 presents the TEA encode routine in C language where the key value is stored in k[0] – k[2] and data are stored in v[0] – v[1]. P.israsena (1)

AES/Rijndael encryption

AES stands for Advanced Encryption Standard. AES is a symmetric key encryption technique which replaces the commonly used Data Encryption Standard (DES). It was the result of a worldwide call for submissions of encryption algorithms issued by the US Government's National Institute of Standards and Technology (NIST) in 1997 and completed in 2000. The winning algorithm, Rijndael, was developed by two Belgian cryptologists, Vincent Rijmen and Joan Daemen. AES provides strong encryption and was selected by NIST as a Federal Information Processing Standard in November 2001 (FIPS-197).[2] The AES algorithm uses three key sizes: a 128-, 192-, or 256-bit encryption key. Each encryption key size causes the algorithm to behave slightly differently, so the increasing key sizes not only offer a larger number of bits with which you can scramble the data, but also increase the complexity of the cipher algorithm.[3]

Data Encryption Standard (DES)

The earliest standard that defines the algorithm (ANS X9.52, published in 1998) describes it as the "Triple Data Encryption Algorithm (TDEA)" — i.e. three operations of the Data Encryption Algorithm specified in ANSI X3.92 — and does not use the terms "Triple DES" or "DES" at all. FIPS PUB 46-3 (1999) defines the "Triple Data Encryption Algorithm (TDEA)", but also uses the terms "DES" and "Triple DES". It uses the terms "Data Encryption Algorithm" and "DES" interchangeably, including starting the specification with. [4]

Data Encryption Standard (DES) is a block cipher with 64-bit block size that uses 56-bit keys. DES was invented over 20 years ago by IBM in response to a public request from the National Bureau of Standards. Due to recent advances in computer technology, some experts no longer consider DES secure against all attacks; since then Triple-DES (3DES) has emerged as a stronger method. Using standard DES encryption, Triple-DES encrypts data three times and uses a different key for at least one of the three passes giving it a cumulative key size of 112-168 bits [2].

2.1.3 Symmetric Encryption vulnerabilities

Breaking symmetric Encryption

There are two methods of breaking symmetric encryption - brute force and cryptanalysis. Brute Force Attack is a form of attack in which each possibility is tried until success is obtained. Typically, a cipher text is deciphered under different keys until plaintext is recognized. No encryption software that is

entirely safe from the brute force method, but if the number of possible keys is high enough, it can make a program astronomically difficult to crack using brute force. But the more bits in a key, the more secure it is, so choose software with as many bits as possible. Cryptanalysis is a form of attack that attacks the characteristics of the algorithm to deduce a specific plaintext or
The key used.

Weak passwords

In every kind of encryption software, there is some kind of password that must be created so that the recipients of the information can read it. Creating a strong password that cannot be easily guessed is just as important as choosing a good algorithm or strong encryption software.

Remembering passwords

If you forget your password, you will not be able to decrypt data that you have encrypted. Be sure to make a backup copy of your password and store
It in a safe place. [22]

Secret keys exchanging and storing

Symmetric key algorithms require sharing the secret key - both the sender and the receiver need the same key to encrypt or decrypt data. Anyone who knows the secret key can decrypt the message. So it is essential that the sender and receiver have a way to exchange secret keys in a secure manner. The weakness of symmetric algorithms is that if the secret key is discovered, all messages can be decrypted.

CHAPTER: - 3 Proposed Approach

3.1Background and Motivation

Background many symmetric block ciphers have been presented in recent years. The Tiny encryption Algorithm (TEA) (Wheeler et al., 1994) is a compromise for safety, ease of implementation, lack of specialized tables, and reasonable performance. TEA can replace 1 Design software, and is short enough to integrate into almost any program on any computer. Some attempts have been made to find weakness of the Tiny Encryption Algorithm.[5] The motivation of this research is to study and implement the proposed attacks on TEA to determine whether such attempts are practically feasible. [2] Tiny Encryption Algorithm the Tiny Encryption Algorithm is a Feistel type cipher (Feistel, 1973) that uses operations from mixed (orthogonal) algebraic groups. A dual shift causes all bits of the data and key to be mixed repeatedly. The key schedule algorithm is simple; the 128-bit key K is split into four 32-bit blocks K = (K [0], K[1], K[2], K[3]). TEA seems to be highly resistant to differential cryptanalysis (Biham et al., 1992) and achieves complete diffusion (where a one bit difference in the plaintext will cause approximately 32 bit differences in the cipher text). Time performance on a workstation is very impressive. [9]

The Tiny Encryption Algorithm is one of the fastest and most efficient cryptographic algorithms in existence. It was developed by David Wheeler and Roger Needham at the Computer Laboratory of Cambridge University. It is a Feistel cipher which uses operations from mixed (orthogonal) algebraic groups – XOR, ADD and SHIFT in this case. This is a very clever way of providing Shannon's twin properties of diffusion and confusion which are necessary for a secure block cipher, without the explicit need for P-boxes and S-boxes respectively. It encrypts 64 data bits at a time using a 128-bit key[14]. It seems highly resistant to differential cryptanalysis, and achieves complete diffusion (where a one bit difference in the plaintext will cause approximately 32 bit differences in the ciphertext) after only six rounds. Performance on a modern desktop computer or workstation is very impressive You can obtain a copy of

Roger Needham and David Wheeler’s original paper describing TEA, from the Security Group ftp site at the world-famous Cambridge Computer Laboratory at Cambridge University. There’s also a paper on extended variants of TEA which addresses a couple of minor weaknesses (irrelevant in almost all real world applications), and introduces a block variant of the algorithm which can be even faster in some circumstances.[17]

Motivation

Motivation As computer systems become more pervasive and complex, security is increasingly important. Cryptographic algorithms and protocols constitute the central component of systems that protect network transmissions and store data. The security of such systems greatly depends on the methods used to manage, establish, and distribute the keys employed by the cryptographic techniques. Even if a cryptographic algorithm is ideal in both theory and implementation, the strength of the algorithm will be rendered useless if the relevant keys are poorly managed. State of the art Cryptography is the art and science behind the principles, means, and methods for keeping messages secure. [2]Cryptanalysis is a study of how to compromise (defeat) cryptographic mechanism. There are two classes of key-based encryption algorithms: symmetric (or secret-key) and asymmetric (or public-key) algorithms. Symmetric algorithms use the same key for encryption and decryption, whereas asymmetric algorithms use different keys for encryption and decryption. Ideally it is infeasible to compute the decryption key from the encryption key. Symmetric algorithms can be divided into stream ciphers and block ciphers. Stream ciphers encrypt a single bit of plain text at a time, whereas block ciphers take a number of bits (say 64 bits), and encrypt them as a single unit. Symmetric encryption is the backbone of many secure communication systems. Dozens of symmetric algorithms have been invented and implemented, both in hardware and software. [6]

Background many symmetric block ciphers have been presented in recent years. The Tiny Encryption Algorithm (TEA) (Wheeler et al., 1994) is a compromise for safety, ease of implementation, lack of specialized tables, and reasonable performance. [16]TEA can replace design software, and is short enough to integrate into almost any program on any computer. Some attempts have been made to find weakness of the Tiny Encryption Algorithm. The motivation of this research is to study and implement the proposed attacks on TEA to determine whether such attempts are practically feasible [9]

3.2 TEA Encryption and Decryption algorithm

Wheeler et al. (1994) at the computer laboratory of Cambridge University developed the TEA encode routine. Presents the TEA encode routine in C language where the key value is stored in k[0] – k[2] and data are stored in v[0] – v[1]. [1]

3.2.1 Encryption Routine

```
void code (long* v, long* k) {

unsigned long y = v[0], z = v[1], sum = 0, /* set up */

delta = 0x9e3779b9, n = 32; /* a key schedule constant */

while (n-->0) {/* basic cycle start */

sum += delta;

y += (z<<4)+k[0] ^ z+sum ^ (z>>5)+k[1] ;

z += (y<<4) +k [2] ^ y+sum ^ (y>>5) +k [3]; /* end cycle */

}

v [0] = y ; v[1] = z ; }
```

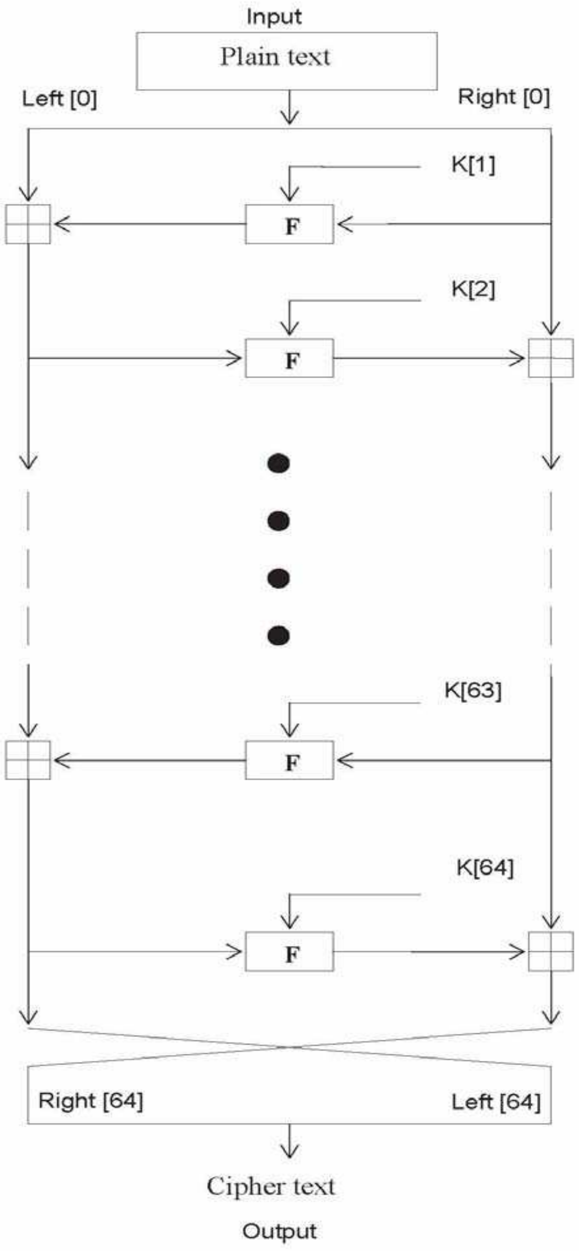


Figure 3.1 The abstract structure of TEA encryption routine.

Figure 3.1 shows the structure of the TEA encryption routine. [1] The inputs to the encryption algorithm are a plaintext block and a key K .The plaintext is P = (Left [0], Right [0]) and the cipher text is C = (Left [64], Right [64]). The plaintext block is split into two halves, Left [0] and Right [0]. Each half is used to encrypt the other half over 64 rounds of processing and then combine to produce the cipher text block [1].

- Each round *i* has inputs Left [*i*-1] and Right [*i*-1], derived from the previous round, as well as a sub key K[*i*] derived from the 128 bit overall K.

- The sub keys K[*i*] are different from K and from each other.
- The constant delta =$_{31}$(51)*2- =, is derived from the golden $_{h}$9E3779B9 number ratio to ensure that the sub keys are distinct and its precise value has no cryptographic significance.

- The round function differs slightly from a classical Fiestel cipher structure in that integer addition modulo 2^{32} is used instead of exclusive-or as the combining Operator.

Figure 3. 2 present the internal details of the *i*th cycle of TEA. The round function, [2] F consists of the key addition, bitwise XOR and left and right shift operation. We can describe the output (Left [*i* +1] , Right[*i* +1]) of the *i*th cycle of TEA with the input (Left[*i*] ,Right[*i*]) as follows [2]

Left [*i*+1] = Left[*i*] F (Right[*i*], K [0, 1], delta[*i*]),

Right [*i* +1] = Right[*i*] F (Right [*i* +1], K [2, 3], delta[*i*]),

delta[*i*] = (*i* +1)/2 * delta,

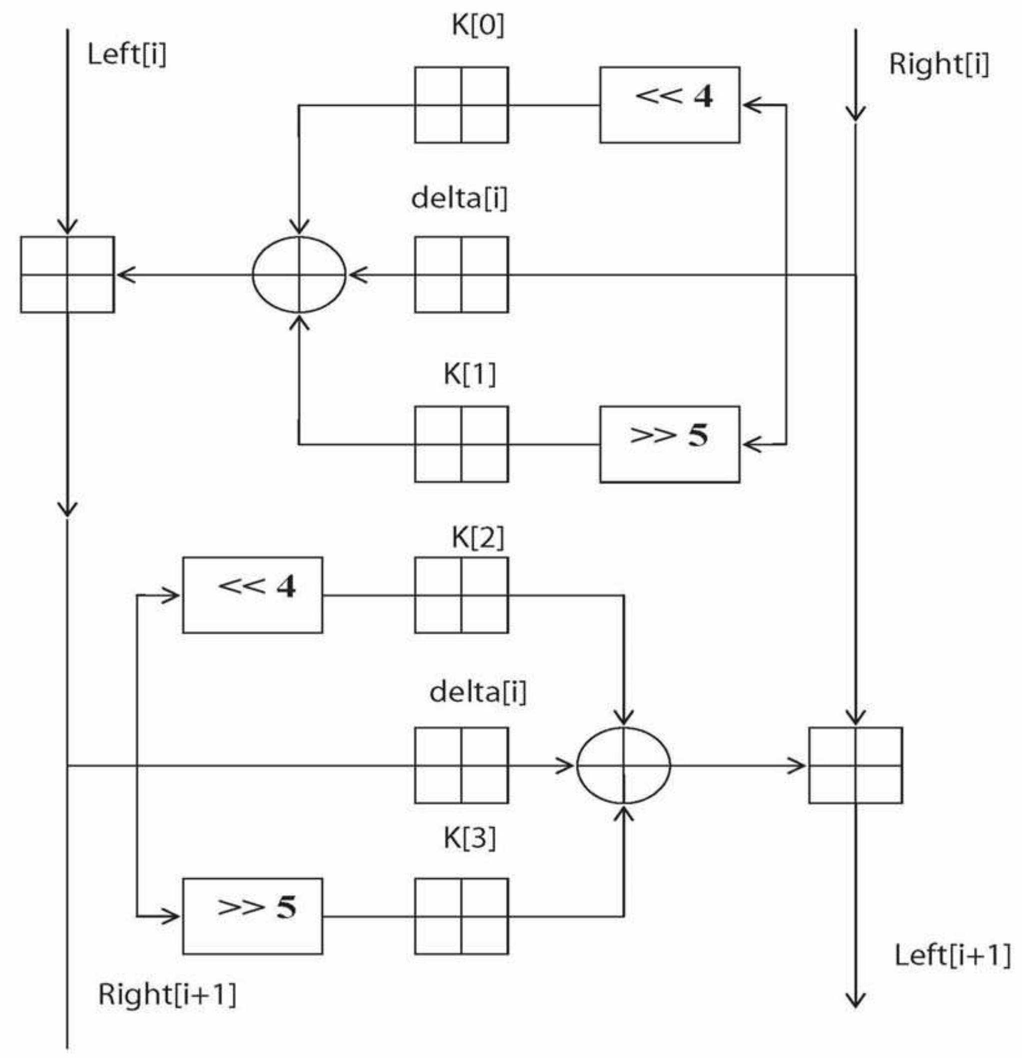


Figure 3.2 An abstraction of *i*-th cycle of TEA

The round function, F, is defined by

F (M, K [*j*,*k*], delta[*i*]) = ((M << 4) K[*j*]) ⊕ (M delta[*i*]) ⊕ ((M >> 5) K[*k*]).

The round function has the same general structure for each round but is parameterized by the round sub key K[*i*]. The key schedule algorithm is simple; the 128-bit key K is split into four 32-bit blocks K = (K[0], K[1], K[2], K[3]).

The keys K[0] and K[1] are used in the odd rounds and the keys K[2] and K[3] are used in even rounds.

3.2.2 Decryption Routine

```
Void decode (long* v, long* k) {
Unsigned long n = 32, sum, y = v[0], z = v[1],
delta = 0x9e3779b9;
sum = delta<<5;
/* start cycle */
while (n-->0) {
z - = (y<<4) +k [2] ^ y+sum ^ (y>>5) +k [3];
y -= (z<<4) +k [0] ^ z+sum ^ (z>>5) +k [1];
sum -= delta;}
/* end cycle */
v [0] = y ; v[1] = z ; }
```

Decryption is essentially the same as the encryption process; in the decode routine the cipher text is used as input to the algorithm, but the sub keys K[*i*] are used in the reverse order.[1]

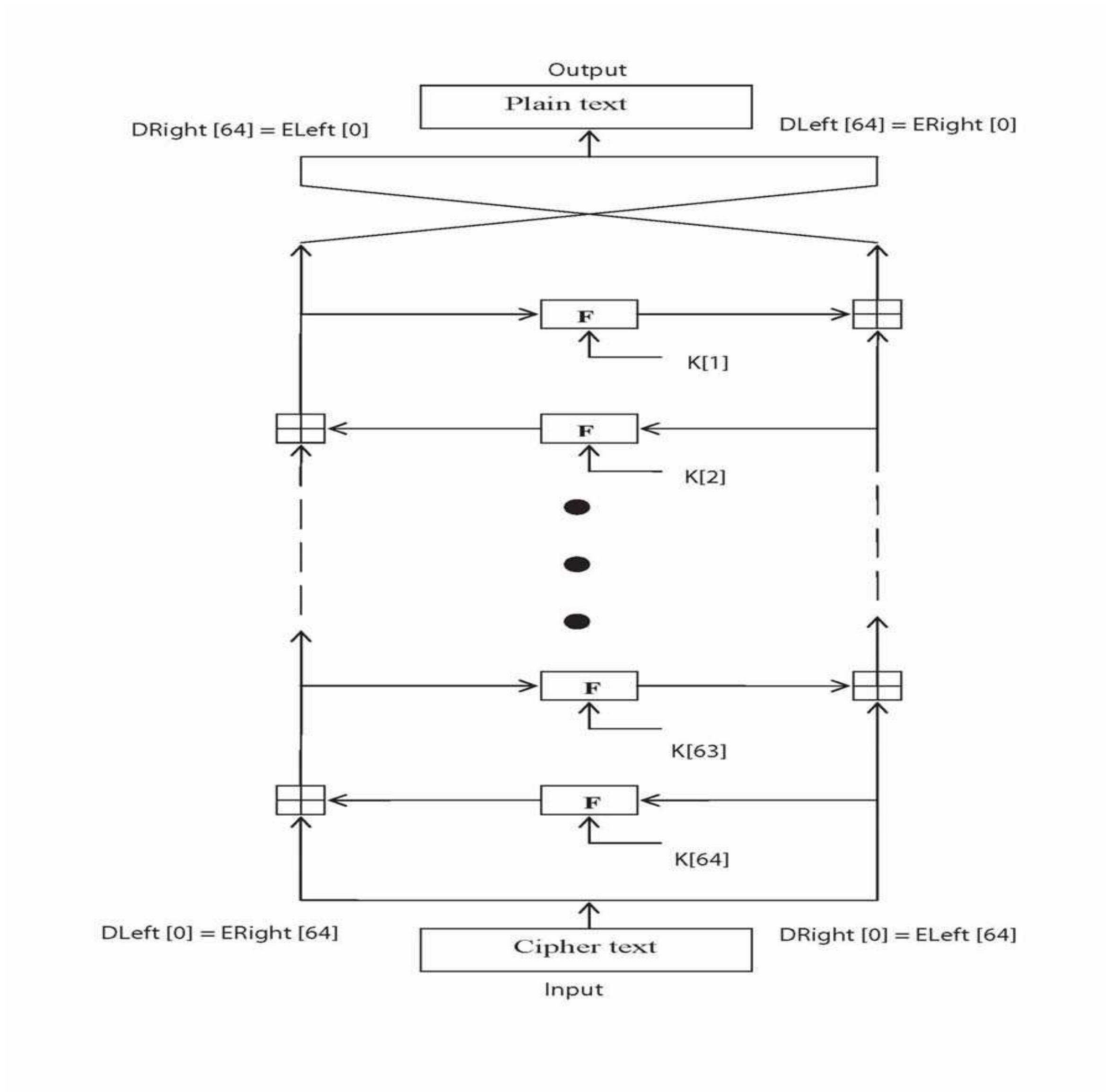


Figure 3.3 The abstract structure of TEA decryption routine.

Figure 3.3. Presents the structure of the TEA decryption routine.[2] The intermediate value of the decryption process is equal to the corresponding value of the encryption process with the two halves of the value swapped.

For example, if the output of the nth encryption round is [2]

ELeft[*i*] || ERight[*i*] (ELeft[*i*] concatenated with ERight[*i*]).

Then the corresponding input to the (64-*i*) th decryption round is

DRight[*i*] || DLeft[*i*] (DRight[*i*] concatenated with DLeft[*i*]).

After the last iteration of the encryption process, the two halves of the output are swapped, so that the cipher text is ERight[64] || ELeft[64], the output of that round is the final cipher text C. Now this cipher text is used as the input to the decryption algorithm. The input to the first round is ERight [64] || ELeft [64], which is equal to the 32-bit swap of the output of the 64th round of the encryption process.

3.2.3 Basics of the routine

It is a Festal type routine although addition and subtraction are used as the reversible operators rather than XOR. The routine relies on the alternate use of XOR and ADD to provide no linearity. A dual shift causes all bits of the key and data to be mixed repeatedly.

The number of rounds before a single bit change of the data or key has spread very close to 32 is at most six, so that sixteen cycles may suffice and we suggest 32. The key is set at 128 bits, as this is enough to prevent simple search techniques being effective.

The top 5 and bottom four bits are probably slightly weaker than the middle bits. These bits are generated from only two versions of z (or y) instead of three, plus the other y or z. Thus the convergence rate to even diffusion is slower. However the shifting evens this out with perhaps a delay of one or two extra cycles.

The key scheduling uses addition, and is applied to the unshifted z rather than the other uses of the key. In some tests k [0] etc. were changed by addition, but this version is simpler and seems as effective. The number delta, derived from the golden number is used where delta = $(\sqrt{5}-1)2^{31}$A different multiple of delta is used in each round so that no bit of the multiple will not change frequently. We suspect the algorithm is not very sensitive to the value of delta and we merely need to avoid a bad value. It will be noted that delta turns out to be odd with truncation or nearest rounding, so no extra precautions are needed to ensure that all the digits of sum change.

The use of multiplication is an effective mixer, but needs shifts anyway. It was about twice as slow per cycle on our implementation and more complicated. The use of a table look up in the cycle was investigated. There is the possibility of a delay ere one entry of the table is used. For example if k [z&] is used instead of k[0], there is a chance one element may not be used of $(3/4)^{32}$, and a much higher chance that the use is delayed appreciably. The table also needed preparation from the key. Large tables were thought to be undesirable due to the set up time and complication. The algorithm will easily translate into assembly code as long as the exclusive or is an operation. The hardware implementation is not difficult, and is of the same order of complexity as DES, taking into account the double length key.

3.2.4 A Security Algorithm TEA

There has been no known successful cryptanalysis of TEA. It's believed to be as secure as the IDEA algorithm, designed by Massey and Xuejia Lai. It uses the same mixed algebraic group's technique as IDEA, but it's very much simpler, hence faster. Also its public domain, whereas IDEA is patented by Ascom-Tech AG in Switzerland. IBM's Don Coppersmith and Massey independently showed that mixing operations from orthogonal algebraic groups performs the diffusion and confusion functions that a traditional block cipher would implement. As a simple plug-in encryption routine, it's great. The code is lightweight and portable enough to be used just about anywhere. It even makes a great random number generator for Monte Carlo simulations and the like. The minor weaknesses identified by David Wagner at Berkeley are unlikely to have any impact in the real world, and you can always implement the new variant TEA which addresses them. If you want a low-overhead end to- end cipher (for real-time data, for example), then TEA fits the bill.

3.3 Proposed work

The TEA can also be used in such open system under special arrangement similar to that proposed by Xingxin Gao[2] a TEA based system may be adopted to implement the hash function at the expense of an increase in overall system complexity, and addition requirement such as RAM.

This thesis is purposed to attempt the implementation of TEA algorithm using secret - key in java core for secure file system with base on cryptography. Java is independent platform.

3.4 Implementation

3.4.1 Requirement to implement new algorithm

The new algorithm (TEA) was implemented using JDK1.4. The algorithm is implemented using java language that's why it can run on any operating system which has java runtime environment.
Implementation uses the design document to produce code. Demonstration that the program satisfies its specifications validates the code. Typically, sample runs of the program demonstrating the behavior for expected data values and boundary values are required. It may take several iterations of the model to produce working program.

Data for encryption and decryption is 64-datablock, and 128 bit keys. Keys which is used have define both in client and server , so that it doesn't need to distributed of key A simple improvement is to copy k[0-3] into a,b,c,d before the iteration so that the indexing is taken out of the loop. In one implementation it reduced the time by about 1/6th.

Java was chosen

The Java platform (JDK 1.4) was used to implement the TEA algorithms. The following are some of the main reasons below.

* Java is considered platform independent because Java compiler produces byte code rather than machine code for a specific type of hardware - this feature of Java makes sure that the programs will run on any platform (with Java interpreter). Thus, the implemented algorithms can be tested on a variety of platforms for comparison purposes.* Java (in particular JDK 1.4) provides a large library of built-in classes and methods (in the form of API) that assist the programmer in writing code for cryptographic algorithms. For example, the BigInteger class in Java lets the programmer apply arithmetic and bit manipulation operations on integer values of arbitrarily large sizes.

* Conversion from integer to string and vice versa, and Likewise conversion of integer values from one radix to another is relatively easier in Java due to the built-in routines provided for this purpose. For instance, one frequently needs to convert a decimal value to binary or hexadecimal, and vice versa.

* The concepts of object serialization and stream input/output make it easy to read and write objects to external disk files however, using Java to implement cryptography algorithms has some drawbacks as well. The main drawback of using Java is its slow speed - this is because Java compiler does not generate native machine code, rather it produces an intermediate form code (called byte code) which needs an interpreter to run. This could have been a concern because the performance of various algorithms had to be tested, but the effect of inefficiency was balanced out because all the algorithms were implemented in the same language (Java) and were tested on the same platform. As mentioned earlier, the primary goal of this research was not to have the most efficient implementation of cryptography algorithms – but just to compare the relative performance of various popular algorithms. So the algorithms were implemented as is, using a uniform language, and were tested on a uniform platform. Each of the above algorithms was implemented as a Java class. In the subsequent

sections, for each algorithm, a description of the instance variables and the implemented methods is provided.

Tests

A few tests were run to detect when a single change had propagated to 32 changes within a small margin. Also some loop tests including a differential loop test to determine loop closures.

A considerable number of small algorithms were tried and the selected one is neither the fastest, nor the shortest but is thought to be the best compromise for safety, ease of implementation, lack of specialized tables, and reasonable performance. On languages, which lack shifts and XOR, it will be difficult to code. Standard C does makes an arithmetic right shift and overflows implementation dependent so that the right shift is logical and y and z are unsigned.

Usage

This type of algorithm can replace DES in software, and is short enough to write into almost any program on any computer. Although speed is not a strong objective with 32 cycles (64 rounds) on one implementation it is three times as fast as a good software implementation of DES which has 16 rounds.

The modes of use of DES are all applicable. The cycle count can readily be varied, or even made part of the key. It is expected that increasing the number of iterations can enhance security.

Analysis

The shifts and XOR cause changes to be propagated left and right, and a single change will have propagated the full word in about 4 iterations. Measurements showed the diffusion was complete at about six iterations.

There was also a cycle test using up to 34 of the bits to find the lengths of the cycles. A more powerful version found the cycle length of the differential function. D(x) =f(x XOR 2^p) XOR f(x) which may test the resistance to some forms of differential crypto-analysis.

Chapter:-4 Conclusion and Future Work

The principal goal guiding the implementation of tiny Encryption Algorithm must be security file system against unauthorized attack .The Tiny Encryption Algorithm is thought to be one of the fastest and most efficient cryptographic algorithms. TEA is a Feistel cipher that uses only XOR, ADD and SHIFT operations to provide Shannon's properties of diffusion and confusion necessary for a secure block cipher without the need for P-boxes and S-boxes. TEA operates on a 64-bit data block using a 128-bit key and can achieve complete diffusion after six rounds. Earlier research's proposed TEA algorithm implement in C language, and hardware encryption core such as radio frequency identification (RFID)design for parallel , serial and digit serial architecture using public –key and microcontrollers in Assembly language.

For software implementation, the java core code is used and portable and therefore particularly suits real-time applications. . Although TEA has a few weaknesses, most notably from equivalent keys and related-key attacks the former is the weakness that led to a method for hacking. This technique is successfully applied over reduce round version of the block cipher TEA. This thesis purposed is based on securing file system so that it can be applicable on applications such as banking and online Transaction processing. At last this thesis is an attempt to implement the TEA algorithm using secret-key in java core for secure file system with base on cryptography.

A proposed direction for the future work could be to analyze the performance /security trade -off of great depth for instance , an algorithm with more complex round and large Number of round is generally consider more secure .the impact of these and other such factor on the overall performance of an algorithm needs to be measured .For future enhancement to this application public key encryption can be applied where two keys can be generated: one to encrypt a file using the public key and another private key to decrypt it. Also, other more advanced encryption operations can be included to enhance the security of the application so that it can be used to encrypt more sensitive administrative material.

Some other software package and different algorithm technique may also be chosen for further study to encryption and decryption of data can be applied for other application such as Digital Signatures’, Mutual Authentication, and Secure data transmission. Adding graphical user interface for file location and file name this project only supports command line.

REFERENCES:

[1] P. Israsena IEEE 2005 “Design and Implementation of Low Power Hardware Encryption for Low Cost Secure RFID Using TEA” page no 1402 to 1406.

[2] Xingxin Gao, Zhe Xiang, Hao Wang, Jun Shen, Jian Huang, and Song ,An approach to security and privacy of RFID system for supply chain,2004 IEEE International Conference on E-Commerce Technology for Dynamic E-Business, pp. 164 - 168, Sept. 2004

[3] Eka Suwartadi, Candra Gunawan, Ary Setijadi P, Carmadi Machbub “ First Step Toward Internet Based Embedded Control System” page 1226-1231. 2005 5TH Asian Control Conference.

[4] S.A Weis, "RFID privacy workshop," IEEE Security & Privacy Magazine, Issue 2, vol.2, pp. 48-50, Mar-Apr 2004

[5] Stephan J Engberg, Morten B Harning, and Christian D Jensen, Zero- Knowledge Device Authentication: Privacy & Security Enhanced RFID preserving Business Value and Consumer Convenience," Proceedings of second annual conference on Privacy,

[6] Security and Trust, pp. 89-101, Brunwick, Canada, October 13-15, 2004.

[7] Philippe Golle, Markus Jacobson, Ari Juel, and Paul Syverson, "Universal Re-encryption for Mixnets," Proc. RSA Conference Cryptographers' Track '04, pp. 163-178, 2004

[8] S. Liu, O.V. Gavrylyako, P.G. Bradford, "Implementing the TEA algorithm on Sensors", ACMSE '04, April 2-3, 2004.

[9] Julio Cesar Hernandez, Pedro Isasi IEEE 2003“Finding efficient distinguishers for cryptographic mappings, with an application to the block cipher TEA page 2189- 2193.

[10] Sozo Inoue and Hiroto Yasuura, "RFID Privacy Using User-controllableniqueness," Proc. RFID Privacy Workshop, MIT, Massachusetts, November 15,2003.

[11] Finkenzeller, RFID-Handbook, 2nd edition -Fundamentals and Applications in Contactless

[12] Moon, D., Hwang, K., Lee, W., Lee, S., & Lim, J. (2002). Impossible differential cryptanalysis of reduced round XTEA and TEA. *In Fast Software Encryption –Proceedings of the 9th International Workshop*

[13] Dukjae Moon, Kyungdeok Hwang, Wonil Lee, Sangjin Lee and Jongim Lim. Impossible Differential Clyptanalysis of Reduced Round XTEA and TEA. Fast Software Encryption, FSE 2002, Leuven, Belgium, February 4-6, 2002. Springer LNCS, v.2365. pp 49-60

[14] F. Akyildiz, W. Su, E. Cayirci, Y. Sankarasubramaniam, *"A Survey on Sensor Networks"*, IEEE communications magazine, Aug 2002.

[15] Steve Bono, Matthew Green, Adam Stubblefield, Ari Juels, Avi Rubin, and Michael Szydlo,"Security Analysis of a Cryptographically-Enabled RFID Device," Draft academic paper http://www.rfidanalysis.org

[16] H. Qi, S. Sitharama Iyengar, K. Chakarabarty, "Distributed sensor networks- a review of recent research", Journal of the Franklin Institute, 2001

[17] Stephen A Weis, Sanjay E Sarma, Ronald L Rivest, and Daniel W Engels, "Security and Privacy Aspects of Low-Cost Radio Frequency Identification Systems," Proc. First International Conference on Security in Pervasive Computing, Boppard, Germany, March 12-14, 2003

[18] John Kelsey, Bruce Schneier, David Wagner Mod **n** cryptoanalysis with applications against RC5P and M6, Proceedings of the 1999 Fast Software Encryption Workshop, pp. 139-155 Springer-Verlag. 1999.

[19] The Complete Reference Java2 by H. Schildt.

[20] http://java.sun.com/j2se

[21] http://javaworld.com

[22] http:// www.cs.utexas.edu/tutorial/index.html

[23] Cryptanalysis of the TINY Encryption Algorithm by VIKRAM REDDY ANDEMTUSCALOOSA, ALABAMA 2003 1-61pages

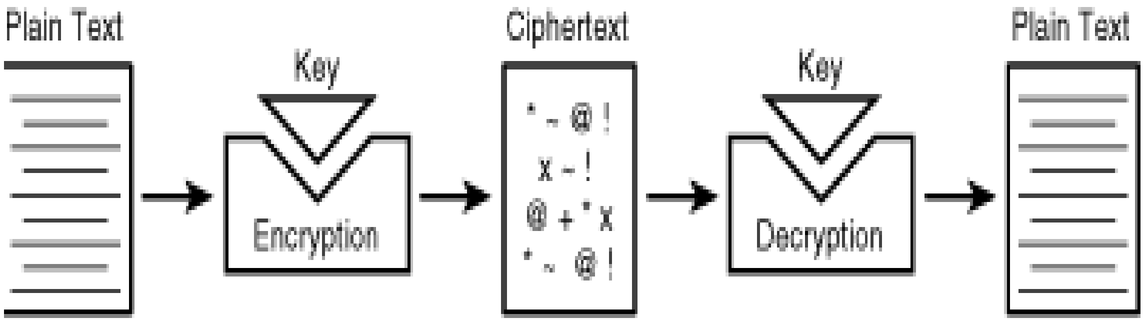
Appedix-1

Data Flow Diagram of File Encryption System

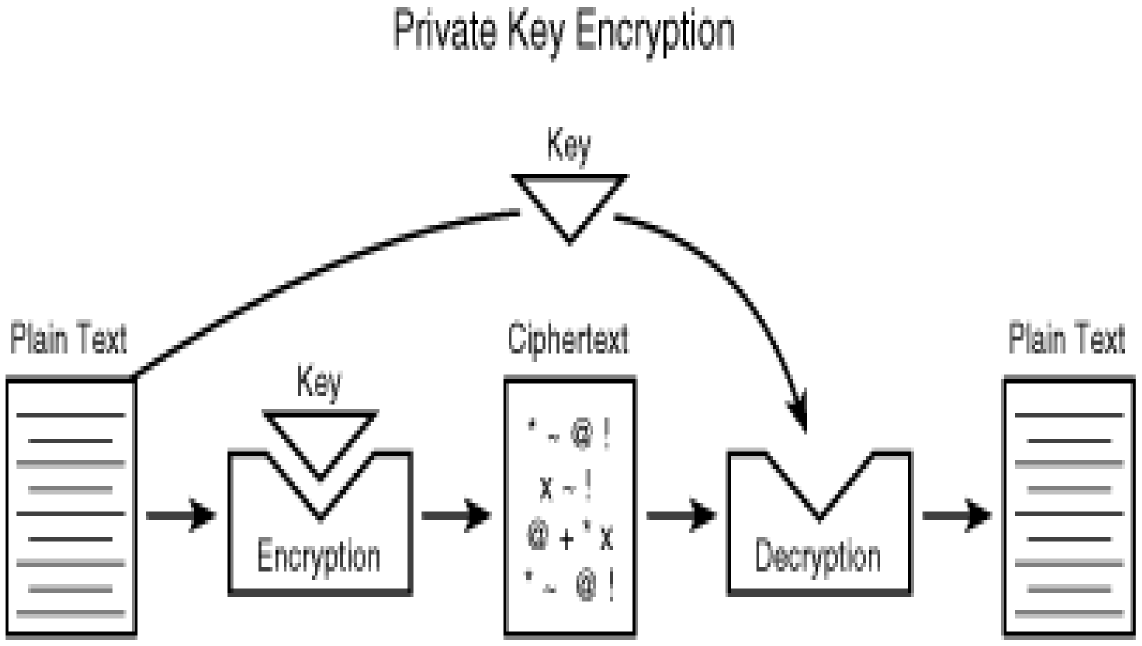
Diagrams for Encryption and Decryption of Data Flow

There are two aspects to consider when planning for transmission security. The first aspect, discussed in the preceding paragraph, is how transmissions are physically sent (that is, over wire or air). The impossibility of preventing physical interception should now be clear. The second aspect of secure transmission relates to the content that is being transmitted. Securing the content of the message is done through encryption.

Encryption involves transforming messages to make them legible only for the intended recipients. *Encryption* is the process of translating *plain text* into *ciphertext*. Human-readable information intended for transmission is plain text, whereas ciphertext is the text that is actually transmitted. At the other end, *decryption* is the process of translating ciphertext back into plain text. *Encryption algorithm* refers to the steps that a personal computer takes to turn plain text into ciphertext. A *key* is a piece of information, usually a number that allows the sender to encode a message only for the receiver. Another key also allows the receiver to decode messages sent to him or her.



Source **[22]** Figure 4.1 Encryption and Decryption of Data



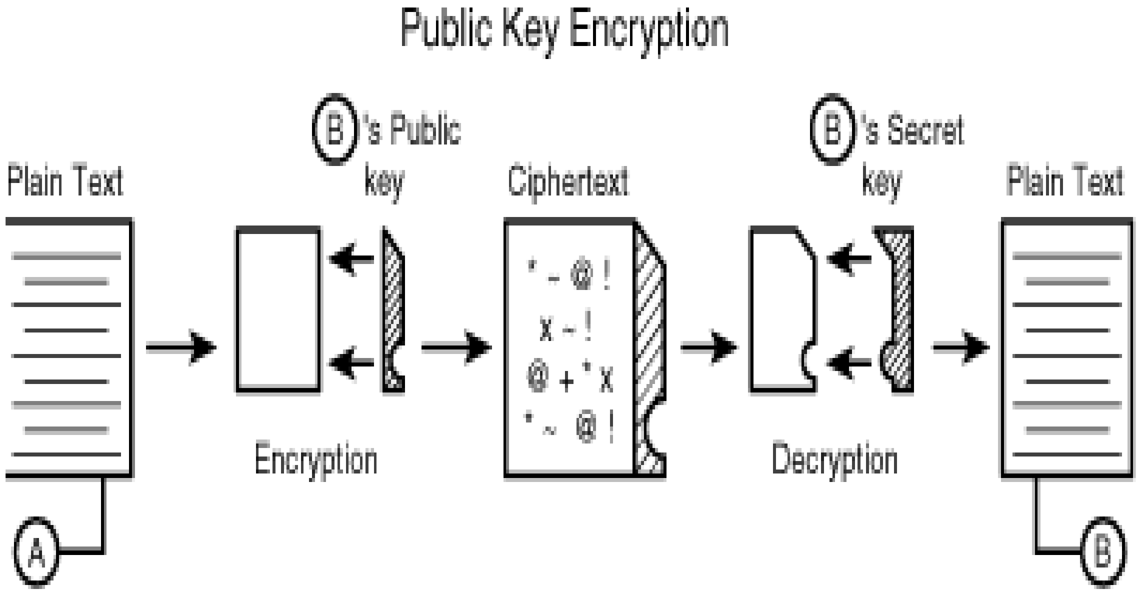


Figure 4.2 Private Key and Public Key Encryption

Appendix -2

Source Code

```
import java.io.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.border.*;
import java.applet.*;
import java.lang.*;

class GUI implements Action Listener

{
            JFrame f;

            JPanel p;

            JTextField tf1;

            JButton b1,b2,b3,b4;

            JTextArea t;

    GUI()

            {
                        f=new JFrame("File Encryption System");

                        p=new JPanel();

                        b1=new JButton("Encrypt");

                         b2=new JButton("Decrypt");

b3=new JButton("Information");

b4=new JButton("Exit");

tf1=new JTextField(20);

 t=new JTextArea(10,30);

 f.getContentPane().add(p);
```

```
p.add(b1);

p.add(b2);

p.add(b3);

p.add(b4);

p.add(tf1);

p.add(t);

b1.addActionListener(this);

b2.addActionListener(this);

b3.addActionListener(this);

b4.addActionListener(this);

f.setVisible(true);

f.setSize(300,300);

}

 public void actionPerformed(ActionEvent ae)

{

tf1.setText(ae.getActionCommand());

if (ae.getSource() == b1)

{

// perform action for button b1

tf1.setText("encryption begin");

byte[] theFile;

String txt = JOptionPane.showInputDialog("Please enter the path of the file: ");


byte[] readFromFile = null;

try
{

FileInputStream in = new FileInputStream(txt);
readFromFile = new byte[in.available()];

in.read(readFromFile);

in.close();

}

catch(IOException e)

{

t.setText("sorry - file not found");

System.out.println("\nSorry - file not found! You might have entered the wrong location\n"+ "\n Do the procedure again \n" + "\n Project Made By Raman Kumar CTA (ME) DCE -07 \n");

System.exit(0);

}

theFile=readFromFile;

String key = JOptionPane.showInputDialog ("Enter your key (the longer the better):");

// This is an update from previous versions
// Decided it would be easier to put
// encryption stuff in to a class
// -----------------------------------

Encryption = new Encryption (theFile,key);

// encrypt file
// -----------

 encryption.encrypt();

// get encrypted file bytes and save it
// -----------------------------------

byte[] to Save=encryption.getFileBytes();

String text = "";

String tx = JOptionPane.showInputDialog("Enter file name: ");

try
```

```
{

FileOutputStream out = new FileOutputStream(tx);

out.write(toSave);

out.close();

}
catch(IOException e)
{

t.setText("Sorry, but there seems to have been a problem" + "saving your file. Perhaps your hard-drive is full\n" + "or the write permissions need to be changed\n");
}

t.setText("\nYour file has been encrypted and saved\n");

 }

else if (ae.getSource() == b2)

{

// perform action for button b2

tf1.setText("decryption begin");

byte[] theFile;

String txx = JOptionPane.showInputDialog ("Please enter the path of the file: ");


byte[] readFromFile = null;

try

{

FileInputStream in = new FileInputStream (txx);

readFromFile = new byte[in. available()];

in.read(readFromFile);

in.close();

}

catch(IOException e)
```

```
{

t.setText("\nSorry - file not found!\n");

System.out.println("\nSorry - file not found! You might have entered the wrong location\n"+ "\n Do the procedure again \n" + "\n Project Made By ‘Raman Kumar, CTA(M.E) DCE. -07 \n");

System.exit(0);

}

theFile=readFromFile;

JPasswordField pf = new JPasswordField("Enter the key: ");

String key = JOptionPane.showInputDialog("Enter the key: ");

Encryption encryption = new Encryption(theFile,key);

encryption.decrypt();

byte[] toSave=encryption.getFileBytes();

String text = "";

String txtt = JOptionPane.showInputDialog("Enter file name: ");

try

{

FileOutputStream out = new FileOutputStream(txtt);

out.write(toSave);

out.close();

 }

catch(IOException e)

{

t.setText("Sorry, but there seems to have been a problem\n" +"saving your file. Perhaps your hard-drive is full\n" +"or the write permissions need to be changed\n");

}
```

```
t.setText("\nYour file has been decrypted and saved\n");

 }

else if (ae.getSource() == b3)

{

tf1.setText ("information regarding encryption algo......");

t.setText ("\n\important info on key choice: \n\n" + "The longer the key, the
better. This program\n" +"implements a key expansion algorithm that given\n"
+"an average length of user-entered key is almost\n" +"analogous to the one-
time pad encryption method\n\n" +"For example: Use key length of 1: 128bit
encryption\n\n"+"Use key length of 2: 256bit encryption\n\n"+ "Use key length
of 8: 1024bit encryption\n\n"+"etc...\n\n\n");

}

else if (ae.getSource() == b4)

{

tf1.setText("choose to exit frm program");

t.setText("Bye Bye!");
System.out.println("\n THANKS FOR USING THIS APPLICATION \n" +
"\nProject Made By Raman Kumar CTA (M.E) DCE -07 \n");

System.exit(0);

}

}

 public static void main(String args[])

{

GUI g = new GUI();

}

}
import java.io.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.border.*;
```

class Encryption

```
/***************/
{
    private String key;
    private char[] keys;
    private byte[] fileBytes;
    private byte[] fileBytez;
    private int pivot;
    private int inter;
    private long alpha;
    private long beta;
    private long gamma;
    private long delta;
    private long sumA;
    private long sumB;
    private long sumC;
    private long sumD;
    private byte[] fileBytesB;
    private int forLevel2;

    public Encryption(byte[] fileBytes,String key)
    {
            this.fileBytes = fileBytes;
            this.key = key;

// Stick the key in to a character array
// This is so the file bytes can be offset with
// the characters as defined by the overall   //algorithm when either encrypting or
decrypting
// -------------------------------------------------

keys = new char[key.length()];

pivot = (int)(fileBytes.length/2);

// These long values are just random bits of  //data junk that are added during the
encryption //process to add to the overall scrambling //capability I got this idea
from the TINY //encryption algorithm.

delta = 0x9e3779b9;

alpha = 0x7f2637c6;

beta  = 0x5d656dc8;

gamma = 0x653654d9;
// Shift the bits slightly (>> and << bitwise  //operators) as determined by
characters in the key
// ---------------------------------------------------

sumA = (long)(alpha >> key.charAt(0));

sumB = (long)(beta << key.charAt(1));

sumC = (long)(gamma >> key.charAt(2));

sumD = (long)(delta >> key.charAt(3));

if (fileBytes.length%5 > 0)

{

inter = (int)((fileBytes.length-1)/5);

}

else inter = (int)(fileBytes.length/5);

// forLevel2 is used in the level2 method
// --------------------------------------

forLevel2 = key.length();

 }

 /***********************************/
 /*   s o m e     m e t h o d s */
/***********************************/

public void setFileBytes(byte[] newBytes)

 {

fileBytes = newBytes;

 }

public byte[] getFileBytes()

{
return fileBytes;
 }

 /************************************/
   /* D o e s   e x a c t l y   w h a t */
   /* i t   s a y s                 */
 /************************************/

public void encrypt()
{

int f = 0;

boolean truth = true;

// Takes user key and makes a bigger one
// for added security
// -----------------------------------

key = keyStream();

keys = new char[key.length()];
for(int c = 0;c<key.length();c++)

{

keys[c] = key.charAt(c);

 }

System.out.println("\nEncrypting\n");

// the outer for loop ensure that the algorithm
// loops round a lot of time, so that
// the file is encrypted mutliple times
// ------------------------------------------


for(int extra = 0;extra<127;extra++)

 {

for(int i = 0;i<fileBytes.length;i = i + keys.length)

{

if (truth == false)

break;

f = 0;

for(int j = i;j<i+keys.length;j++)

 {
```

```
if(j>=fileBytes.length)

 {

truth = false;

break;

}

fileBytes[j] = (byte)((fileBytes[j] ^(keys[f] - 'A' << sumD)) ^ (keys[f] + sumD));

sumD - = delta;

f++;
 }

}

fileBytes = splitNSwap(fileBytes);

setFileBytes(fileBytes);

 }

setFileBytes(level2(fileBytes,true));

 }
public void decrypt()

{

setFileBytes(level2(fileBytes,false));

int f = 0;

boolean truth = true;

key = keyStream();

keys = new char[key.length()];

for(int c = 0;c<key.length();c++)

{

keys[c] = key.charAt(c);

 }

System.out.println("\nDecrypting\n");
```

```
for(int extra = 0;extra<127;extra++)

 {

fileBytes = getFileBytes();

fileBytes = splitNSwap(fileBytes);
for(int i = 0;i<fileBytes.length;i = i + keys.length)

{

if (truth == false)

 break;

f = 0;

for(int j = i;j<i+keys.length;j++)

{

if(j>=fileBytes.length)

 {

truth = false;

break;

 }

fileBytes[j] = (byte)((fileBytes[j] ^(keys[f] - 'A' << sumD)) ^ (keys[f] + sumD));

sumD - = delta;

f++;

 }

}

setFileBytes(fileBytes);

}

}

 // To add to the confusion, this method basically
// takes the byte[] array as encrypted so far
// splits it in half and then swaps two halves  //around i.e. a b c d e f would become d e f a // b c
// -------------------------------------------------

public byte[] splitNSwap(byte[] zeBytes)

{

if(zeBytes.length%2==0)

{

pivot = (int)(zeBytes.length/2);

}

else pivot = (int)((zeBytes.length-1)/2);

fileBytez = new byte[zeBytes.length];

for(int reverse = 0;reverse<pivot;reverse++)

{

fileBytez[reverse] = (byte)(zeBytes[reverse+pivot]^fileBytez[reverse]);

}

for (int reverseB = pivot;reverseB<zeBytes.length;reverseB++)

{

fileBytez[reverseB] = (byte)(zeBytes[reverseB - pivot]^fileBytez[reverseB]);

 }

setFileBytes(fileBytez);

return fileBytez;

}

 // if a long key is used and only some of //the key is correct then first part of
cipher text //ll still be decrypted so:
// (if you want to see the key that it produces //comment print statement)
// ---------------------------------------------------//-------------



public String scrambleKey(String toBeScrambledFurther)

{
```

```
pivot = (int)(toBeScrambledFurther.length()/2);

String newKey = "";

String sub1 = "", sub2 = "";

for (int a = 0; a<pivot;a++)

{

sub1 += toBeScrambledFurther.charAt(a+pivot);

 }

for (int b = pivot; b<toBeScrambledFurther.length();b++)

{

sub2 += toBeScrambledFurther.charAt(b-pivot);

}

newKey = sub1+sub2;

//System.out.println(newkey);

return newKey;

 }



    // What is this function passed ?
   // Answer - basically two parameters - the first is
  // an array of bytes that need to be scrambled. //The second is a boolean - true if
the    array is //being scrambled and false if being //descrambled etc.....
 // ----------------------------------------------------

public byte[] level2(byte[] oldBytes, boolean state)

{

if(state)

System.out.println ("Scrambling encrypted data");

else System.out.println("\nDescrambling encrypted data");

int s = forLevel2;
int stop = oldBytes.length%s;

byte[] newBytes = new byte[oldBytes.length];

byte[] tempBytes = new byte[oldBytes.length-stop];

byte[] resultBytes = new byte[oldBytes.length];

byte[] remainderBytes = new byte[stop];

int hello = oldBytes.length-stop;

for (int old = 0;old<oldBytes.length-stop;old++)

 {

tempBytes[old] = oldBytes[old];

 }
for (int old = 0;old<stop;old++)

{

remainderBytes[old] = oldBytes[(oldBytes.length-stop+old)];

 }

if (state)

{

for (int outer = 0;outer<s;outer++)

 {

for (int c = outer;c<hello+outer;c+=s)

{

if(c+s<oldBytes.length)

 {

newBytes[c] = (byte)(oldBytes[c+s]-sumA);

newBytes[c+s] = (byte)(oldBytes[c]+sumB);

 }
else break;

  }

  }

 }

else if (!state)

{

for (int outer = s-1;outer >=0;outer--)

 {

for(int c = (hello-1-outer);c>=0-outer;c-=s)

{

if(c-s>=0)

{

newBytes[c-s] = (byte)(oldBytes[c]-sumB);

newBytes[c] = (byte)(oldBytes[c-s]+sumA);

 }

else break;

}

if (outer <= 0)break;else continue;
 }

 }

for(int rep = 0;rep<newBytes.length;rep++)

{

resultBytes[rep]=newBytes[rep];
```

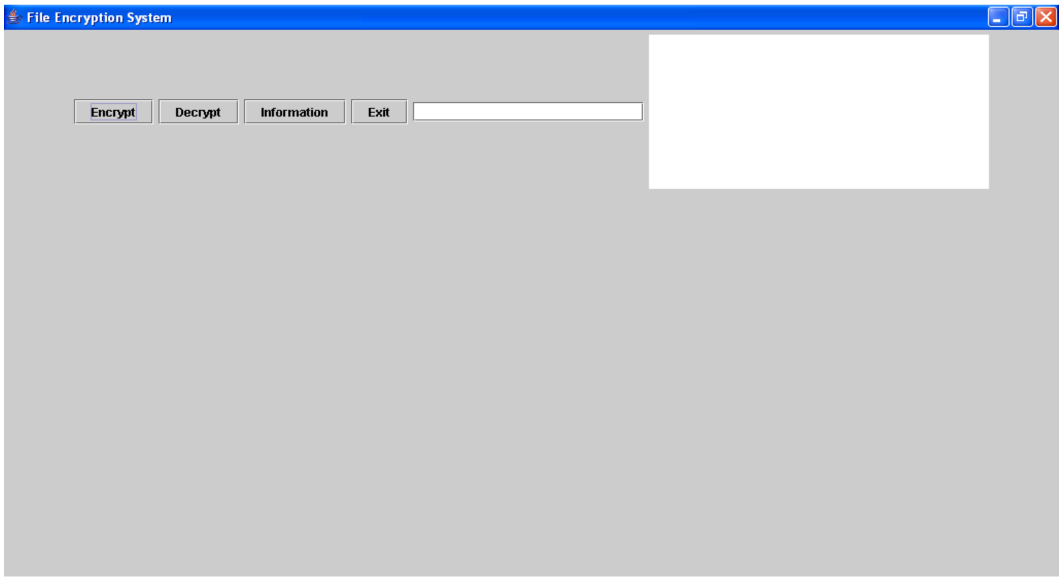
```
}

for(int rep = 0;rep<remainderBytes.length;rep++)

{

resultBytes[rep]=remainderBytes[rep];

 }

setFileBytes(newBytes);

return newBytes;

}
      /* The keyStream() method takes the user key
    * and enlarges it to (key.length()*key.length()+
   * key.length())*128, in the following algorithmic method.
   * This improves security. I.e. longer keys are
   * much harder to crack
   */
 public String keyStream()

{

System.out.println("\nGenerating key stream\n");

String answer = key;

String thekey = key;

for(int i = 0;i<(thekey.length()*128);i++)

 {

answer = answer + getPart(thekey);

thekey = getPart(thekey);

 }

//answer = scrambleKey(answer);
```

```
return answer;

}

 // KeyStream helper method
// -----------------------

public String getPart(String thekey)

{

char[] keyPart = new char[thekey.length()];

String result = "";

for(int c = 0;c<thekey.length()-1;c++)

{

keyPart[c] = (char)(thekey.charAt(c+1) - 1);

}

keyPart[thekey.length()-1] = thekey.charAt(0);

for(int put = 0;put<keyPart.length;put++)

 {

result = result + keyPart[put];

 }
return result;

 }

}
```

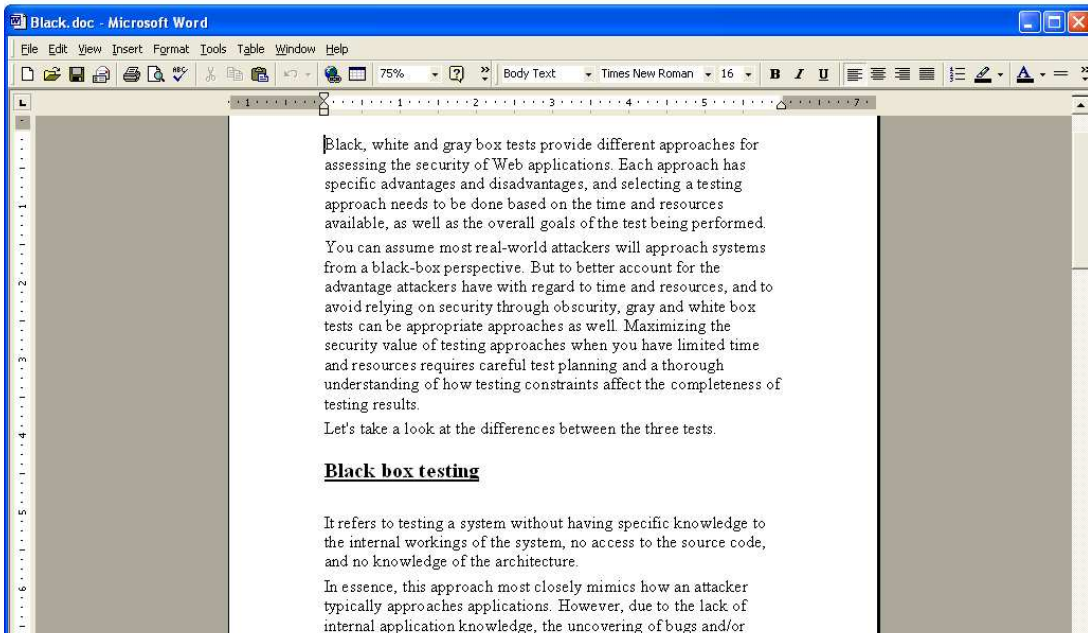
Appendix-3

Result and Snapshot

This is the Frame, which prompts the user when he compiles and runs the program.



This is the file, which is to be encrypted.


Black.doc - Microsoft Word
File Edit View Insert Format Tools Table Window Help
75%
Body Text
Times New Roman
16

Black, white and gray box tests provide different approaches for assessing the security of Web applications. Each approach has specific advantages and disadvantages, and selecting a testing approach needs to be done based on the time and resources available, as well as the overall goals of the test being performed.

You can assume most real-world attackers will approach systems from a black-box perspective. But to better account for the advantage attackers have with regard to time and resources, and to avoid relying on security through obscurity, gray and white box tests can be appropriate approaches as well. Maximizing the security value of testing approaches when you have limited time and resources requires careful test planning and a thorough understanding of how testing constraints affect the completeness of testing results.

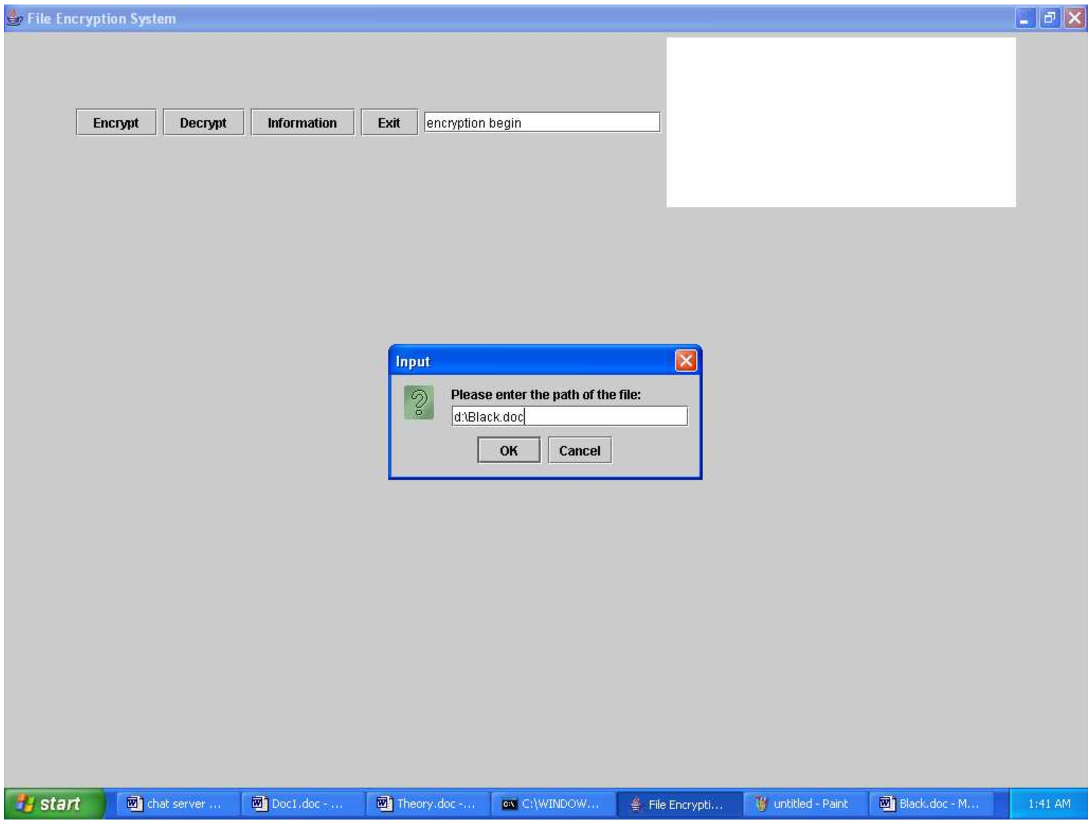
Let's take a look at the differences between the three tests.

Black box testing

It refers to testing a system without having specific knowledge to the internal workings of the system, no access to the source code, and no knowledge of the architecture.

In essence, this approach most closely mimics how an attacker typically approaches applications. However, due to the lack of internal application knowledge, the uncovering of bugs and/or

After pressing the "Encrypt" button, it will make u enter the path of the file.

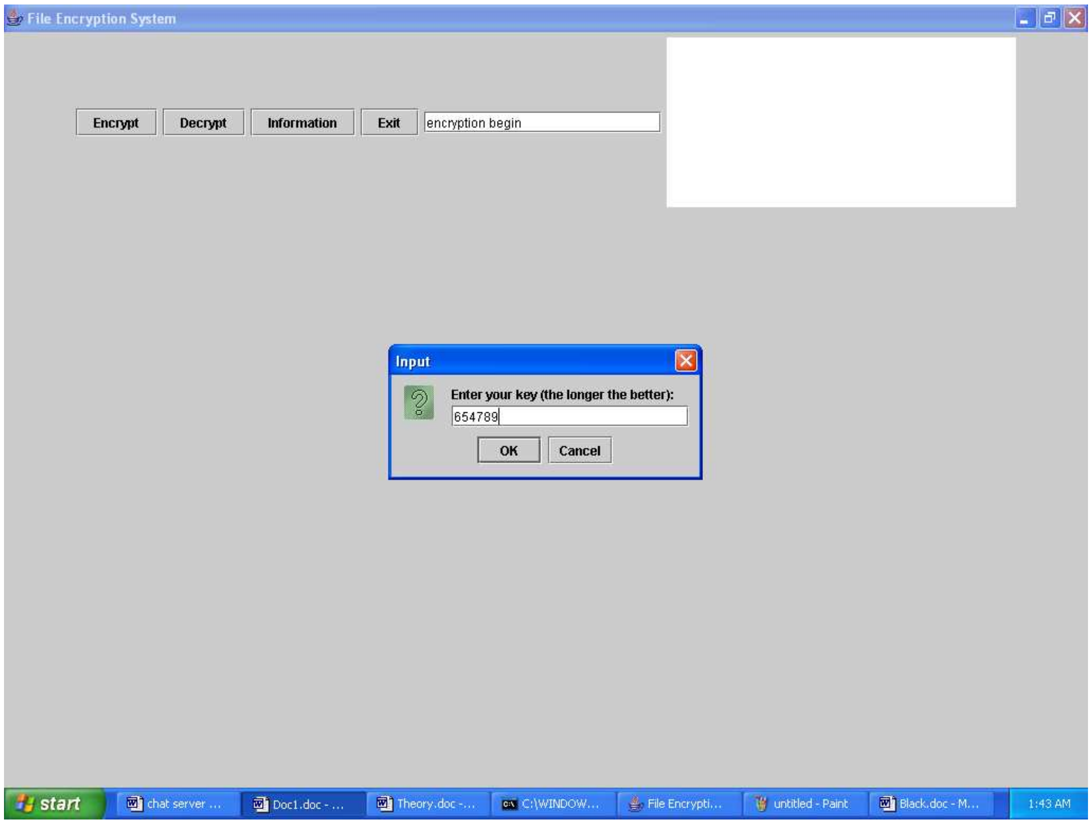


encryption begin

Please enter the path of the file:

d:\Black.doc

Enter the key…. the longer the better.

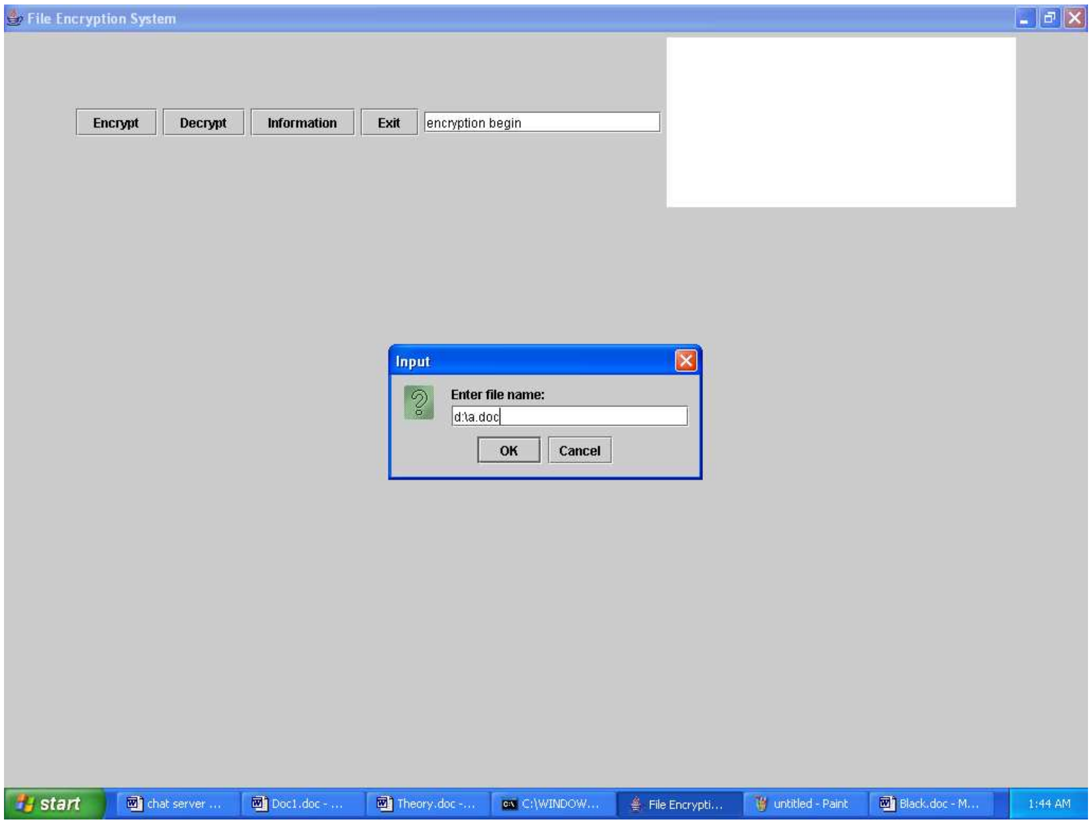


encryption begin

Enter your key (the longer the better):

654789

Enter the file name…the name by which u wants to save the encrypted file.

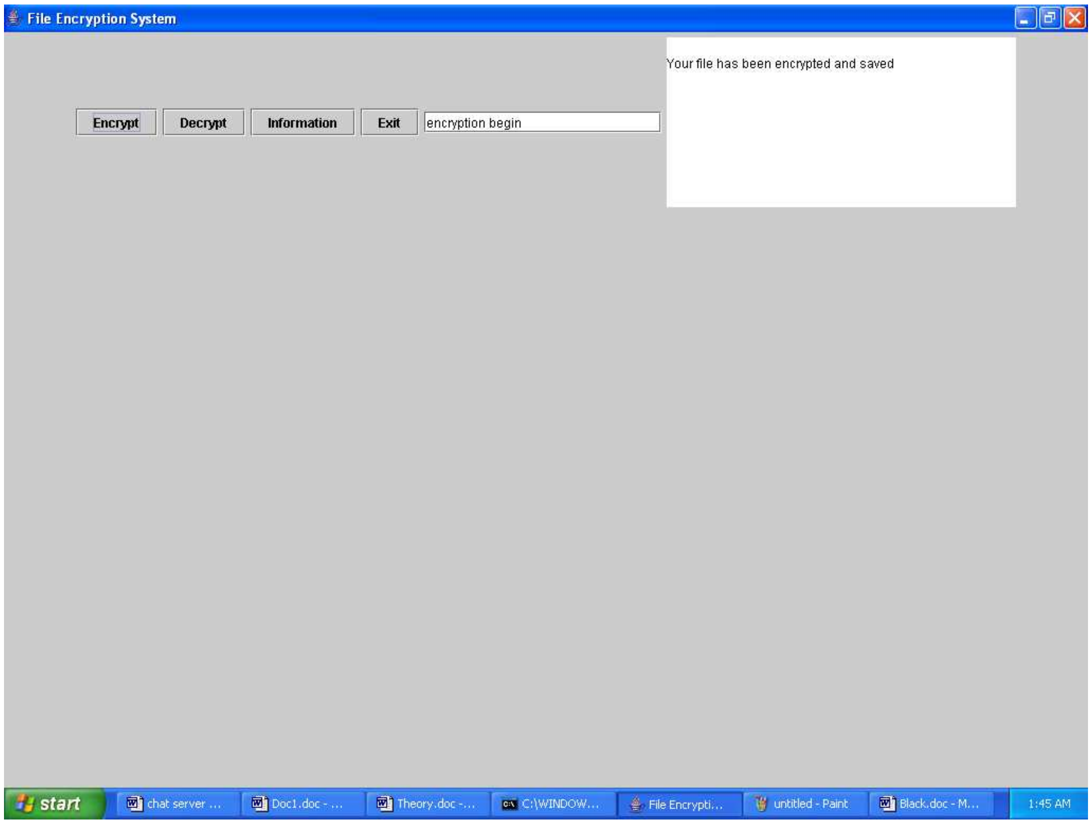


encryption begin

Enter file name:

d:\a.doc

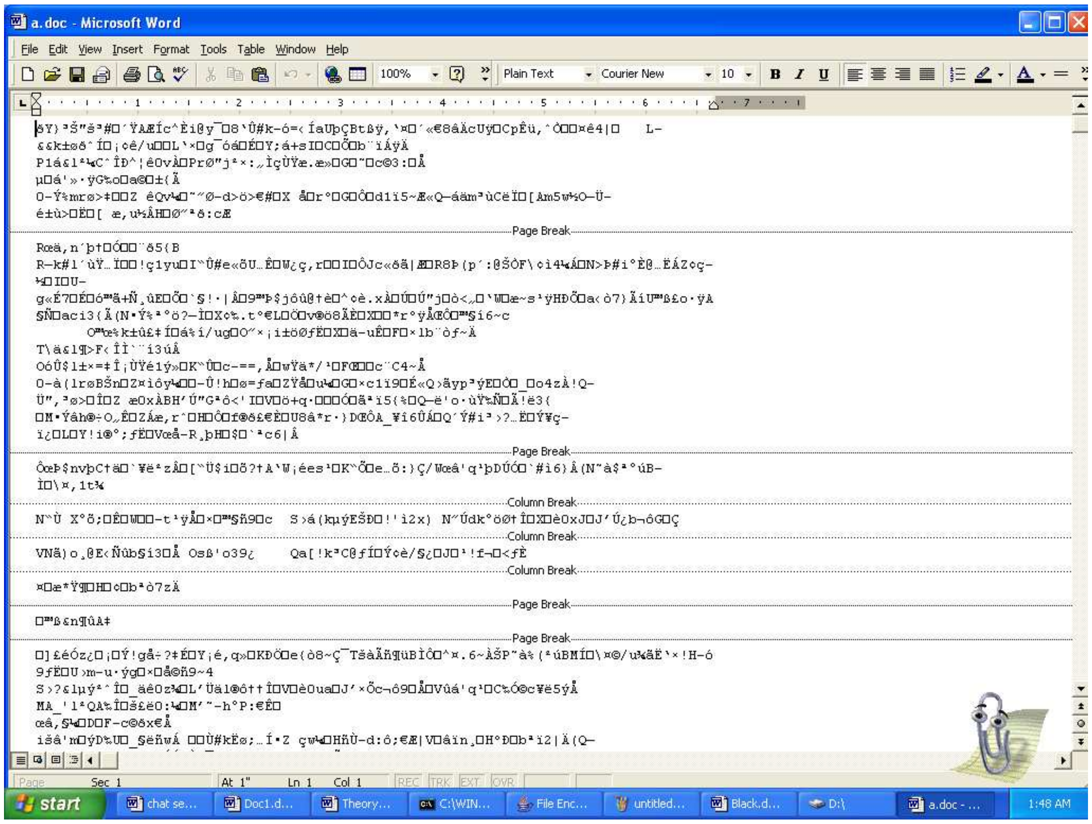
After the Encryption procedure…A message prints in the Text Area.



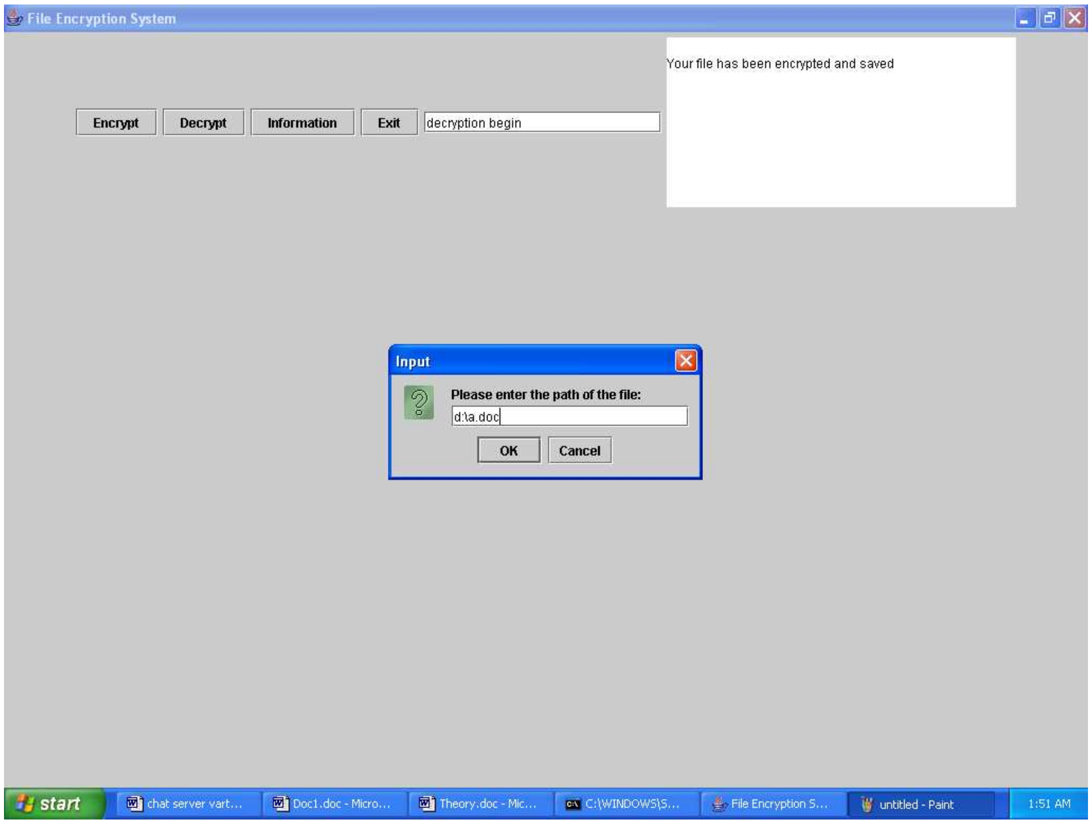
encryption begin

Your file has been encrypted and saved

Encrypted File.



After pressing the “Decrypt” button, it will make u enter the path of the file, which is to be decrypted.



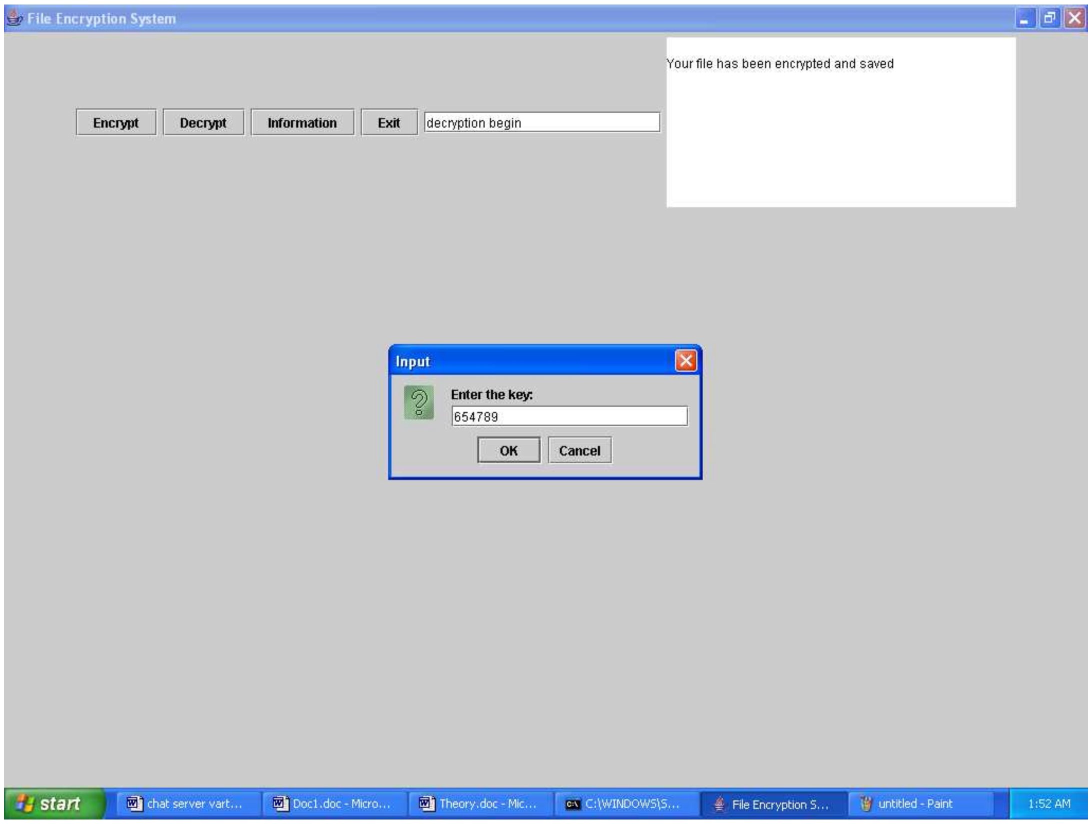
Your file has been encrypted and saved

decryption begin

Please enter the path of the file:

d:\a.doc

Enter the key… the same key which u have entered during the Encrypt procedure.



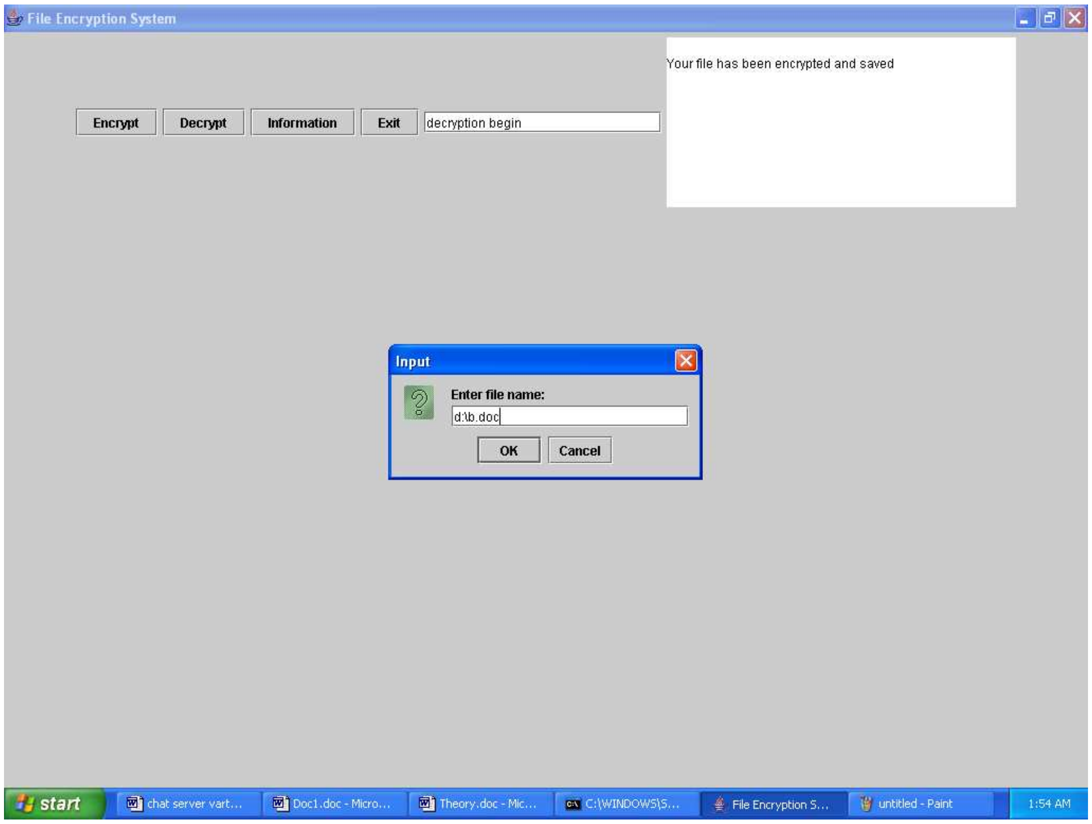
Your file has been encrypted and saved

decryption begin

Enter the key:

654789

Enter the file name…the names by which u want to save the decrypted file.



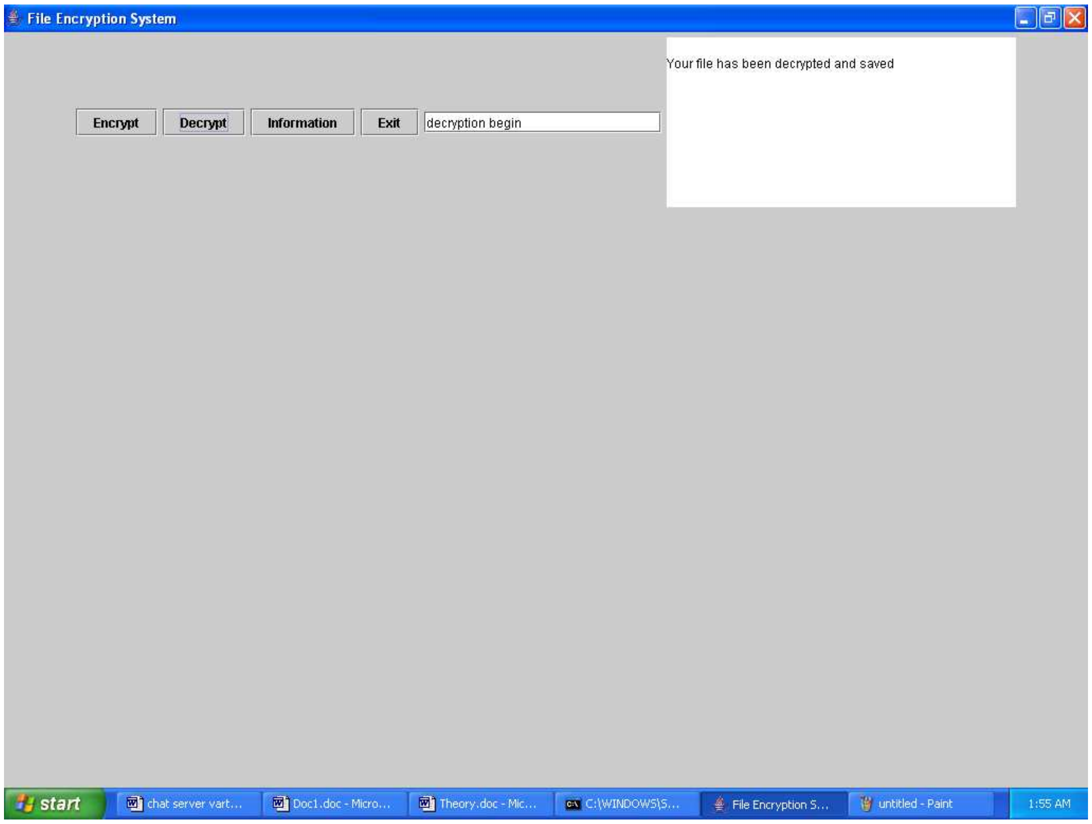
decryption begin

Your file has been encrypted and saved

Enter file name:

d:\b.doc

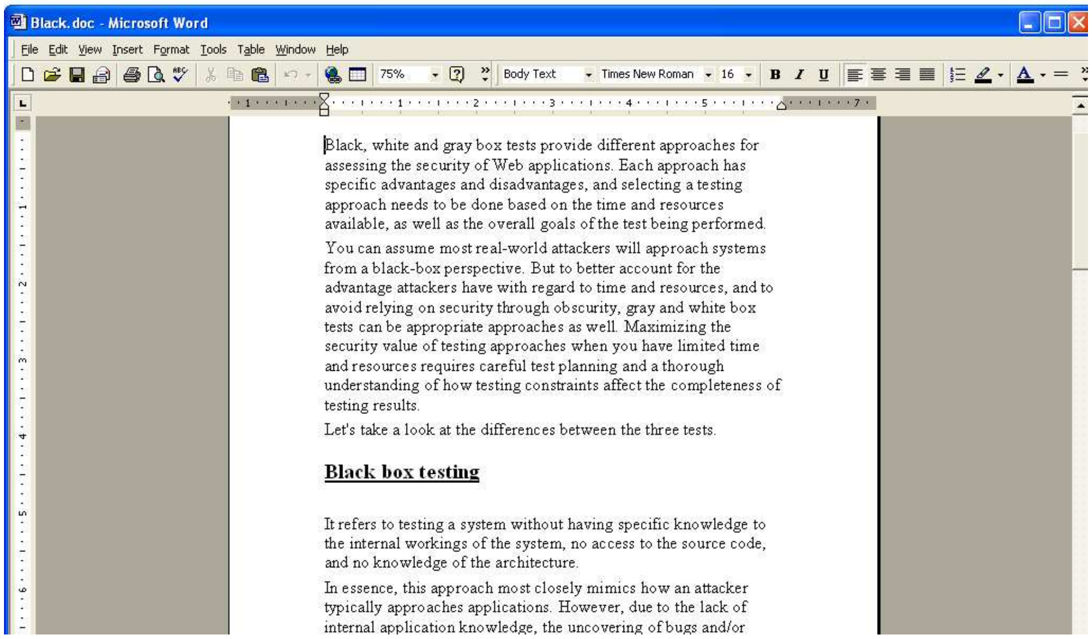
After the Decryption procedure…a message prints in the Text Area.



```
decryption begin
```

```
Your file has been decrypted and saved
```

Decrypted File.


Black.doc - Microsoft Word
File Edit View Insert Format Tools Table Window Help
75%
Body Text
Times New Roman
16

Black, white and gray box tests provide different approaches for assessing the security of Web applications. Each approach has specific advantages and disadvantages, and selecting a testing approach needs to be done based on the time and resources available, as well as the overall goals of the test being performed.

You can assume most real-world attackers will approach systems from a black-box perspective. But to better account for the advantage attackers have with regard to time and resources, and to avoid relying on security through obscurity, gray and white box tests can be appropriate approaches as well. Maximizing the security value of testing approaches when you have limited time and resources requires careful test planning and a thorough understanding of how testing constraints affect the completeness of testing results.

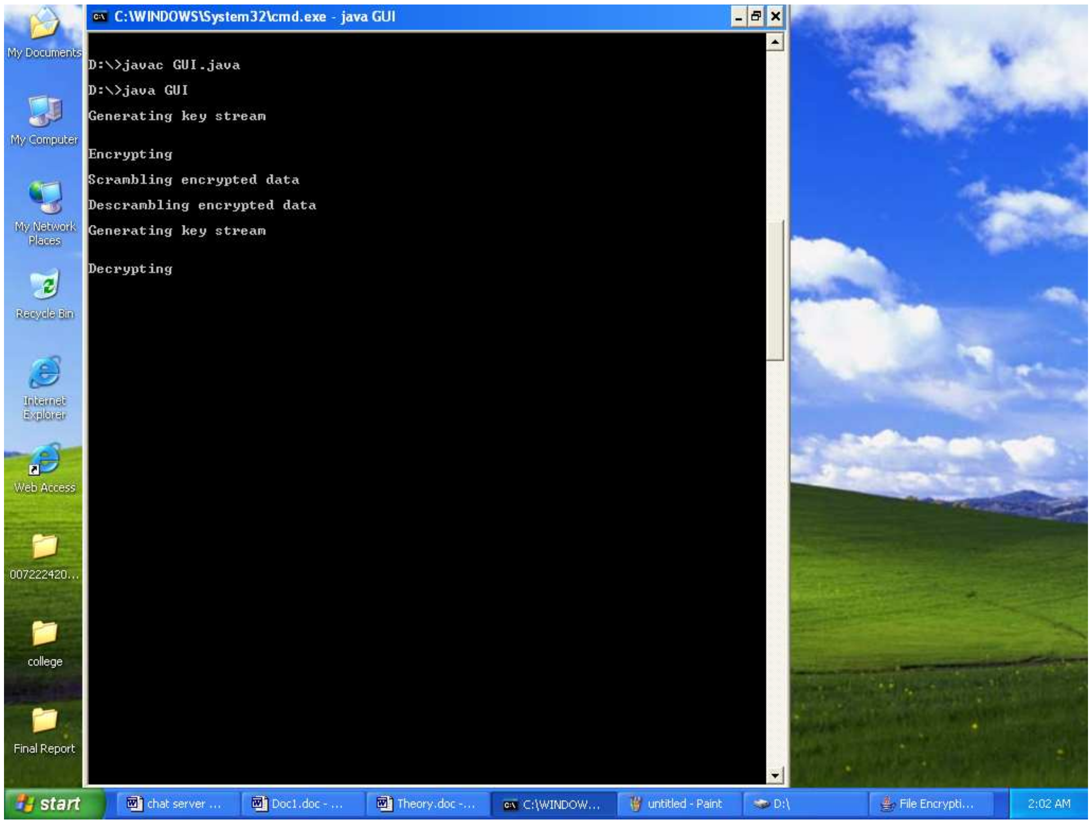
Let's take a look at the differences between the three tests.

Black box testing

It refers to testing a system without having specific knowledge to the internal workings of the system, no access to the source code, and no knowledge of the architecture.

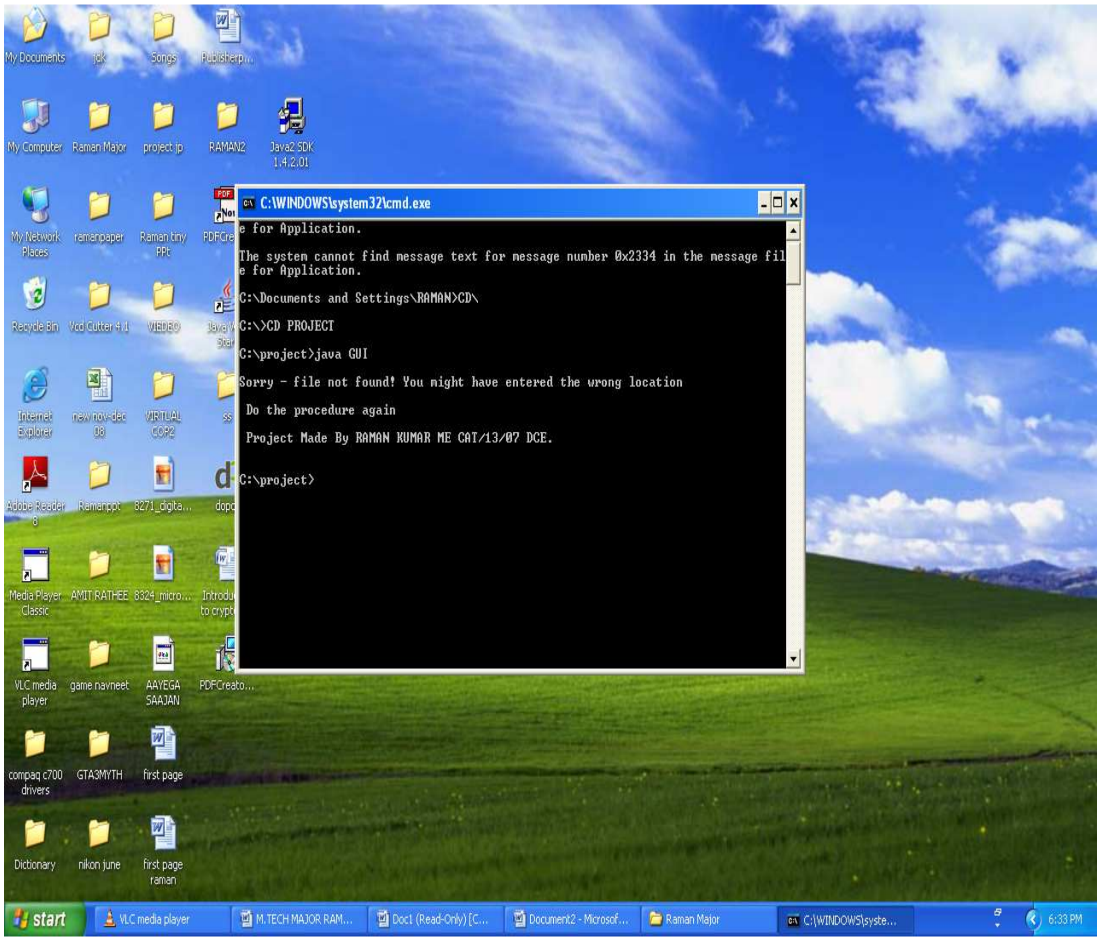
In essence, this approach most closely mimics how an attacker typically approaches applications. However, due to the lack of internal application knowledge, the uncovering of bugs and/or

Following messages appear in the command prompt during the encryption and decryption procedures.



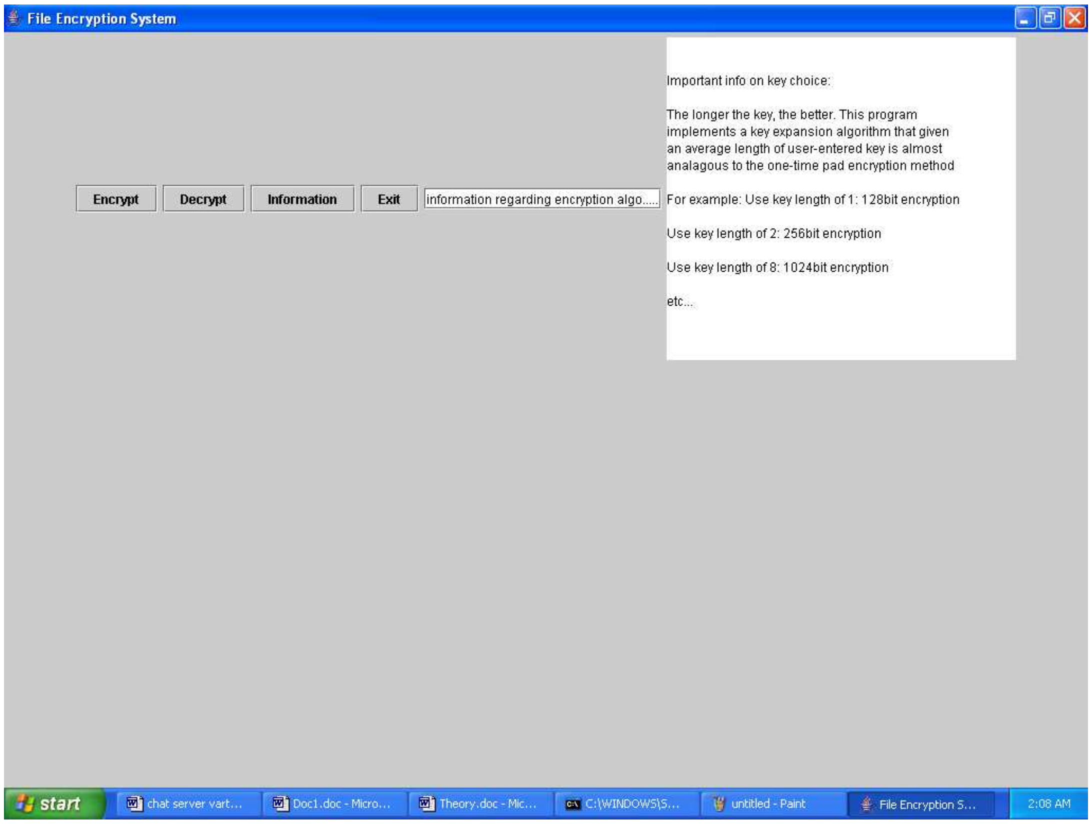
```
D:\>javac GUI.java
D:\>java GUI
Generating key stream

Encrypting
Scrambling encrypted data
Descrambling encrypted data
Generating key stream

Decrypting
```

If file is not found at the entered location, following messages appear in the command prompt during the encryption and decryption procedures.



```
e for Application.

The system cannot find message text for message number 0x2334 in the message fil
e for Application.

C:\Documents and Settings\RAMAN>CD\

C:\>CD PROJECT

C:\project>java GUI

Sorry - file not found! You might have entered the wrong location

 Do the procedure again

 Project Made By RAMAN KUMAR ME CAT/13/07 DCE.


C:\project>
```

When “Information” Button is pressed…. a message prints in the Text Area.



Important info on key choice:

The longer the key, the better. This program
implements a key expansion algorithm that given
an average length of user-entered key is almost
analagous to the one-time pad encryption method

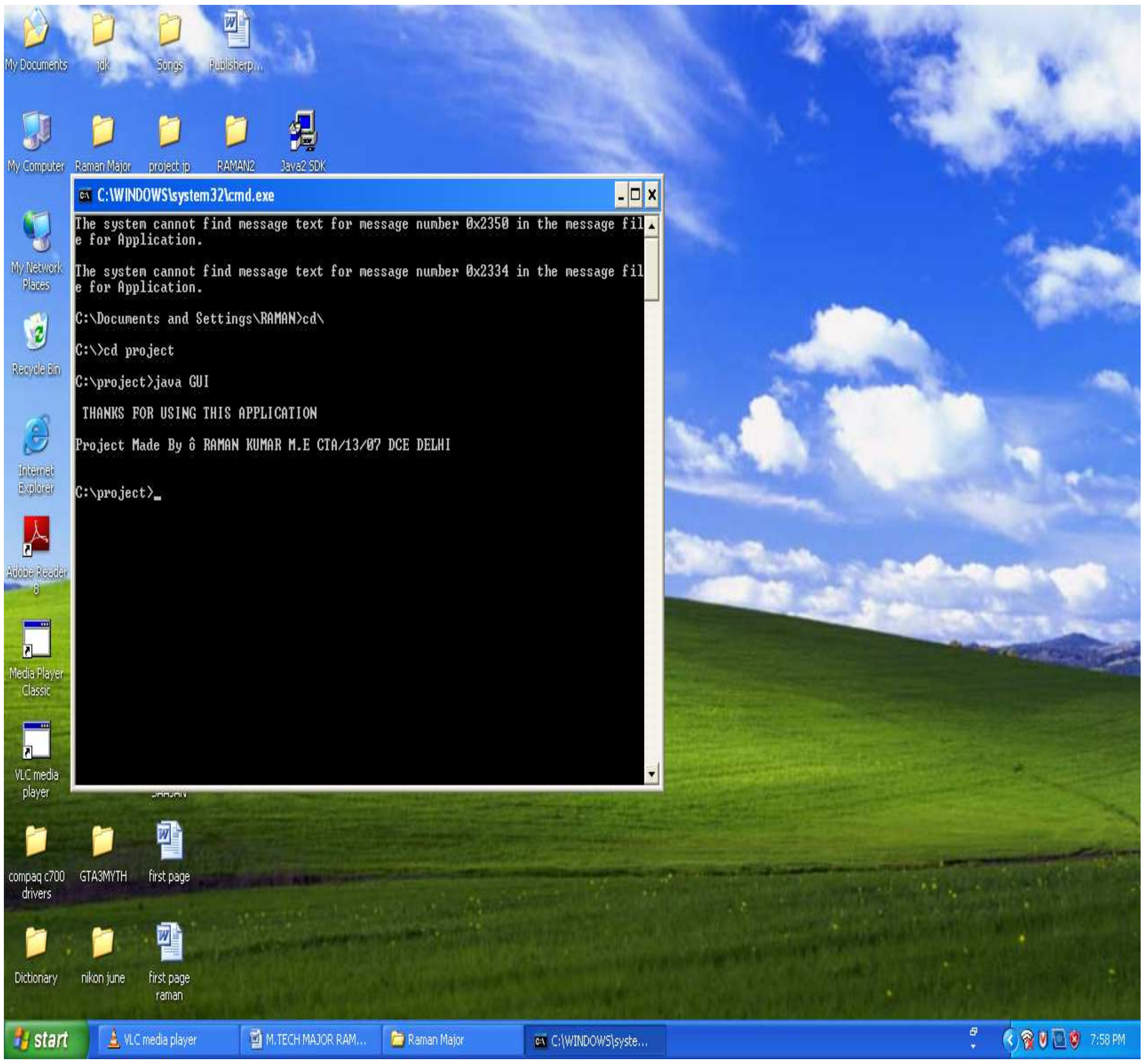
For example: Use key length of 1: 128bit encryption

Use key length of 2: 256bit encryption

Use key length of 8: 1024bit encryption

etc...

When “Exit” button is pressed, following messages appear in the command prompt.



```
The system cannot find message text for message number 0x2350 in the message fil
e for Application.

The system cannot find message text for message number 0x2334 in the message fil
e for Application.

C:\Documents and Settings\RAMAN>cd\

C:\>cd project

C:\project>java GUI

 THANKS FOR USING THIS APPLICATION

Project Made By ô RAMAN KUMAR M.E CTA/13/07 DCE DELHI


C:\project>_
```