DEVELOPMENT OF A TWO-FISH BASED FILE ENCRYPTION AND DECRYPTION SYSTEM

BY

LAWANSON Olubode Francis

PG/SCI/0011

SUBMITTED TO THE DEPARTMENT OF PHYSICAL AND MATHEMATICAL SCIENCES, COLLEGE OF SCIENCE, AFE BABALOLA UNIVERSITY, ADO-EKITI

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARDS OF POST GRADUATE DIPLOMA (PGD) IN COMPUTER SCIENCE.

DECEMBER, 2020

CERTIFICATION

This is to certify that this dissertation, " Development of a Two-Fish Based File Encryption and Decryption System," was written by LAWANSON Olubode Francis (19/PGC/SCI01/1011). It has been examined and approved as meeting part of the requirements for the award of Master's Degree, MSc. in Computer Science.

………………………………………..	…….……………….
MRS. O.A. BABALOLA	Date
(Supervisor)	
………………………………………..	…….……………….
MRS. T OKE	Date
(Co-Supervisor)	
………………………………………..	…….……………….
DR. O.M AWEH	Date
(Programme PG Coordinator)	
………………………………………..	…….……………….
DR. O.A ADEYEMO	Date
(Head of Department)	
………………………………………..	…….……………….
DR	Date
(Provost, College of Science)	
………………………………………..	…….……………….
PROF	Date
(Provost, College of Postgraduate Studies)	

DEDICATION

This work is dedicated to the almighty God, the alpha and omega, the one who was and forever will be, the one who has never let me down in all my endeavors and has continued to crown my effort with success

ACKNOWLEDGEMENTS

My foremost thanks go to the Almighty God for making it possible for the commencement and completeness of this work and for making this research success; all glory belongs to Him. A special thanks go to my hardworking supervisors, Mrs. Babalola, O. A., and Mrs. Tope Oke, for their relentless effort, kindness, and constructive criticism, which have helped to make this work a great success. I appreciate the Head of the Department, Dr. O.A Adeyemo, and the Professors, lecturers, and laboratory staff of the Department of Computer Science. My gratitude also goes to my parents, MR and MRS Lawanson, for their support, encouragement, prayers, love, and financing for this research work. Many thanks to my sisters and brother, who have contributed one way or the other to the success of this work. I am grateful to the Postgraduate Coordinator, Computer Science, Dr. O.M Aweh, for being very thoughtful and willing to listen to us and proffering suggestions whenever needed; your presence and help were instrumental in making this thesis writing process a seamless one. To my lecturers, Dr. Mrs. Abiola, Dr. Mrs. Olatunji, Mrs. Tope Oke, Mrs. Babalola, who taught, nurtured, and impacted my knowledge in computer science throughout my Programme thank you for the knowledge passed on to me, it would come in handy in my future endeavors for that reason I am grateful.

ABSTRACT

Nowadays, information security is becoming more important in data storage and transmission. Attacks and security breaches are increasingly being witnessed by individuals and organizations. Activities of hackers on the internet has led to leakage of classified documents and files, theft of identity and sensitive personal information among others are common occurrence to individuals and organizations alike. This necessitates the need for measures to guarantee personal safety in networked environments. In this project, a file encryption and decryption system is developed for securing personal files either for storage or transmission. The system implements the TwoFish encryption and decryption algorithm, which presents a better performance in terms of speed when compared with other symmetric ciphers such as Lucifer, DES, Triple-DES, and BlowFish without compromising security. The system was implemented in Java programming language using the NetBeans IDE. The development follows the waterfall process model. The developed application can provide additional layer of security to personal files stored on a system or transmitted over a network.

TABLE OF CONTENT

Title Page		i
Certification		ii
Dedication		iii
Acknowledgement		iv
Abstract		v
Table of Content		vi
List of Figures		ix
CHAPTER ONE	INTRODUCTION	
1.1	Background of the Study	1
1.2	Statement of the Problem	2
1.3	Aim and Objectives	3
1.4	Research Methodology	3
1.5	Expected Contribution to Knowledge	3
1.6	Definition of Terms	4
1.7	Organization of Project Work	4
CHAPTER TWO	LITERATURE REVIEW	
2.1	Introduction	6

2.2 What is Cryptography? 7

2.3 Data Encryption/Decryption 7

2.4 Modern Historical Overview 8

2.5 Principles of Modern Cryptography 9

2.6 Symmetric and Asymmetric Algorithms 9

2.7 Ciphers 11

2.8 Types of Cryptographic Algorithms 15

2.9 Cryptanalysis 18

2.10 Types of Cryptographic Attacks 18

2.11 Linear Cryptanalysis and Differential Cryptanalysis 20

CHAPTER THREE SYSTEM DESIGN

3.1 Introduction 22

3.2 Twofish Algorithm 22

3.3 Proposed Architecture 24

3.4 Data Flow Diagram of the System 25

CHAPTER FOUR SYSTEM IMPLEMENTATION

4.1 Introduction 28

4.2 System Requirements 28

4.3 Choice of Programming Language 29

4.4 Deployment Procedure 32

CHAPTER FIVE CONCLUSION AND RECOMMENDATION

5.1 Conclusion 39

5.2 Recommendation 39

5.3 Future Work 40

REFERENCES 41

LIST OF FIGURES

Figure 2.1: Basic Encryption and decryption 8

Figure 2.2: Symmetric Algorithms 10

Figure 2.3: Asymmetric Algorithms 11

Fig 2.4: Transposition cipher 13

Figure 3.2: Twofish Algorithm 24

Figure 3.3: Architecture of the proposed system 25

Figure 3.5: Data Flow Diagram Level 0 26

Figure 3.6: Data Flow Diagram Level 1 27

Figure 4.1: Extracted Application Folder for File Security using Twofish Algorithm 33

Figure 4.2: Application Splash Screen 34

Figure 4.3: Application Home Page 35

Figure 4.4: About Page 36

Figure 4.5: File Encryption Interface 37

Figure 4.6: File Decryption Interface 38

CHAPTER ONE

INTRODUCTION

1.1 BACKGROUND OF THE STUDY

Information and Communication technology (ICT) has attained a level of maturity such that virtually every sector is now driven by ICT in one way or the other. The evolution of computers comes with the evolution of connectivity and inter-connectivity, with early systems connected via local area network, which later evolved to wireless connectivity, and then the internet (Yin, Stecke, and Li, 2018). The internet first started as the inter-connectivity of few desktop computers, which later grew into millions, saw the introduction of laptops and personal digital assistants (PDA), and now, mobile phones. Recent advances in interconnectivity has led to the Internet of Things (IoT) where billions of devices are connected and each generates, processes, stores, and transmits such data (Yaqoob, *et al.,* 2017). These offer numerous benefits to end-users which include wider audience coverage for business, cross-border interaction among various sectors, high processing speed, cheaper cost of storage, improved accessibility to vital information, among others (Duncan and Whittington, 2017). All these benefits however come at a cost of which breaches in privacy, confidentiality, integrity and accessibility tops the concerns in the field of information security (Yaqoob, *et al.,* 2017).

While the field of information security keeps proposing new methodologies for ensuring security while using ICT resources, there exist some of its concepts that are conflicting. In order words, satisfying one implies breaching the other (Qadir and Quadri, 2016). Example of such is Access Control auditing and user privacy.

The use of the internet or a network facility gives a user access to resources stored on various nodes within the network but also exposes the user's system to the wider public irrespective of their location. Although there are different mechanisms of securing oneself on the internet such as the use of strong antivirus program, firewall, as well as different layers of communication encryption protocols, it has been said that no system can be 100% secure. Hence the need for securing personal files on storage media (Pushpa, 2020).

The problem of security becomes more pronounced in developing nations such as Nigeria where most users cannot afford licensed version of software, hence often opt for freeware or cracked version which might have been laced with various malicious programs.

1.2 STATEMENT OF THE PROBLEM

Most system users store sensitive files on system in plain formats. Unauthorized access to these files could compromise the integrity of such file, and also compromise the privacy of the users. Although there are existing applications that have been developed for file encryption, this research focus on the exploration of the Twofish encryption algorithm due to its relatively higher encryption and decryption speed and larger block-size.

The outbreak of COVID-19 has reduced physical interaction among people thereby promoting the use of ICT tools across sector which automatically exposes users to possibility of intrusion. Most users do not have basic knowledge on how to stay secure while using the internet while other have weak firewall configuration, no antivirus program or having outdated or obsolete antivirus program installed. Some have even download virus laden freeware in the name of performance optimization application. This virus-laden software often steals personal information and uploads to a remote server for exploit.

Most system users rely only on password for protection which only prevents unauthorized logon into the system while it has no way to protect the system content. Hence the need for a cost effective solution for personal information management on computer system.

1.3 AIM AND OBJECTIVES

The aim of the project is to develop a secure system for multimedia local file storage for system users. To achieve this goal, the following objectives will be pursued

i. to develop a block-cipher encryption framework for local file encryption using Twofish algorithm
ii. to develop a key management mechanism for easy key storage and retrieval
iii. to implement a file encryption system based on (i) and (ii)

1.4 RESEARCH METHODOLOGY

This research is initiated by surveying various approaches for file security on personal computer using various cryptographic algorithm, which include their strength and weaknesses. A proposed framework was then presented for file encryption using the Twofish encryption algorithm. The application receives as input a file to be encrypted, which could be a text, audio, video, image, or even executable file and the encryption key. The application information, such as file name and file extension were then extracted from the input file. The extracted information is then converted into bits and appended to the file before encryption. The encryption key is expected to be a 128bit string. However, due to the limitation of system users to propose strong passwords for access control, the application converts the string supplied by the user into a 256 bit fixed-length string by hashing. The hashed password is then divided into two part. The left chunk is then used for the encryption. The encryption process is carried out be breaking down the file into byte chunks, encrypting each chunk using the transformed encryption key.

1.5 EXPECTED CONTRIBUTION TO KNOWLEDGE

In this project, an encryption system, that relies on Twofish algorithm, for personal multimedia file encryption on personal computer will be developed. The system will provide an additional layer of security for personal files on computer systems.

1.6 DEFINITION OF TERMS

Some terms commonly used in the context of message security are discussed below:

i. ***Plaintext***: The original message (or data) that the sender wants to sends to the receiver.

ii. ***Cipher text*****:** The scrambled message generated from the plaintext and the key, applied to the encryption algorithm. Each combination of unique message and key will generate a different cipher text.

iii. ***Key*****:** The Shared secret knowledge between the two communicating ends to enable the intended receiver to recover the original message from the scrambled one it receives from the communication channel.

iv. ***Encryption Algorithm:*** Substitutions and computations at sender end to scramble the message.

v. ***Decryption Algorithm:*** The reverse computations at the receiver end to recover the original message.

1.7 ORGANIZATION OF PROJECT WORK

This project is organized into chapters. Chapter 1 presents the background to security in ICT and the need for personal security. It also presents the aim of the project as well as the approach adopted to address the problem identified. Chapter 2 presents the survey of related literatures. The architecture of Twofish algorithm and the implementation framework were presented in Chapter 3. Chapter 4 presents the implementation tools as well as the implementation details for the architecture proposed in the project. Conclusion and recommendation were presented in the last chapter.

CHAPTER TWO

LITERATURE REVIEW

2.1 INTRODUCTION

Cryptography deals with the process of encryption and decryption of data using different techniques in order to guaranty confidentiality of data, being one of the three pillars of information security. The concept of cryptography has been dates back to centuries ago with various modification and improvement proposed over the ages. One of the first forms of cryptography was used by Julius Creaser, and it is commonly known as the Creaser cipher which was a simple substitution cipher. Modern day cryptography, however, requires much more complicated algorithms in order to serve the purpose of data integrity.

The word '***cryptography***' has basically evolved from two words of Greek, 'Crypto' means hidden or secret, while 'Graphein' means to write. Cryptography was traditionally meant for the purpose of confidentiality. It used ***encryption*** to convert plain text to incomprehensible form at the sending end, before the message was submitted for transmission to an insecure shared channel. On the other hand, at the receiving end, the message was recovered from this apparent jumble of meaningless characters using shared secret knowledge, this was called ***decryption***. This was supposed to make the message safe from intruders and eavesdroppers, who would not be able to understand it without the necessary key. In modern contexts, cryptography is a well-developed science which utilizes techniques of Mathematics, Computer Science, and Communication Engineering. Cryptography systems are classified by three different criteria. The first criterion of classifying cryptography algorithms is, the principle used in the encryption algorithm, which may be by substitution and transposition. Most product systems use multiple stages of both. The second criterion is the number of keys being used in the system. If the same key is being used by both sender as well as the receiver, the system is called “Symmetric”. On the other hand, if they use separate keys, then it is called public key, or “Two Key Encryption”. The third criterion is the way in which the processing of plaintext takes place. Block cipher processes blocks of input at a time, while the stream cipher process the input elements on a continuous basis.

2.2 WHAT IS CRYPTOGRAPHY?

Cryptography, to most people, is concerned with keeping communications private. Indeed, the protection of sensitive communications has been the emphasis of cryptography throughout much of its history. If you want to keep information secret, you have two possible strategies: hide the existence of the information, or make the information unintelligible. Cryptography is the art and science of keeping information secure from unintended audiences, of encrypting it. Conversely, cryptanalysis is the art and science of breaking encoded data. The branch of mathematics encompassing both cryptography and cryptanalysis is cryptology Simpson (1997).

2.3 DATA ENCRYPTION/DECRYPTION

Encryption is the transformation of data into some unreadable form. Its purpose is to ensure privacy by keeping the information hidden from anyone for whom it is not intended, even those who can see the encrypted data. Decryption is the reverse of encryption; it is the transformation of encrypted data back into some intelligible form. Encryption and decryption require the use of some secret information, usually referred to as a key. Depending on the encryption mechanism used, the same key might be used for both encryption and decryption, while for other mechanisms, the keys used for encryption and decryption might be different. (x5.net, 2012).

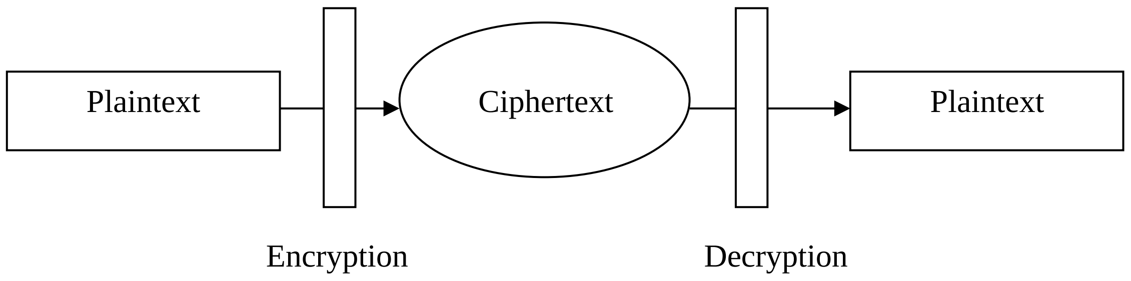


Figure 2.1: Basic Encryption and decryption

Modern cryptography uses sophisticated mathematical equations (algorithms) and secret keys to encrypt and decrypt data. Today, cryptography is used to provide secrecy and integrity to our data, and both authentication and anonymity to our communications.

2.4 MODERN HISTORICAL OVERVIEW

Cryptology was a public field in the United States until World War I, when the Army & Navy realized its value to national security and began working in secret. Through the early 1970s, cryptology was dominated by the government both because computers were very expensive and because the government released very little information. It returned to mainstream academic and scientific communities in a sort of cryptology renaissance when the computer revolution made computers more readily available and when demand for encryption increased due to fundamental changes in the ways America communicated (x5.net, 2012). The increase in demand for cryptography was driven by industry interest (e.g., financial services required secure electronic transactions and businesses needed to secure trade secrets stored on computers), and individual interest (e.g., secure wireless communications). Digital communications were obvious candidates for encryption.

2.5 PRINCIPLES OF MODERN CRYPTOGRAPHY

Modern cryptographers emphasize that security should not depend on the secrecy of the encryption method (or algorithm), only the secrecy of the keys. The secret keys must not be revealed when plaintext and ciphertext are compared, and no person should have knowledge of the key. Modern algorithms are based on mathematically difficult problems - for example, prime number factorization, discrete logarithms, etc. There is no mathematical proof that these problems are in fact are hard, just empirical evidence.

Modern cryptographic algorithms are too complex to be executed by humans. Today's algorithms are executed by computers or specialized hardware devices, and in most cases are implemented in computer software.

The design of secure systems using encryption techniques focuses mainly on the protection of (secret) keys. Keys can be protected either by encrypting them under other keys or by protecting them physically, while the algorithm used to encrypt the data is made public and subjected to intense scrutiny. When cryptographers hit on an effective method of encryption (a cipher), they can patent it as intellectual property and earn royalties when their method is used in commercial products. In the current open environment, many good cryptographic algorithms are available in major bookstores, libraries and on the Internet, or patent office (x5.net, 2012).

2.6 SYMMETRIC AND ASYMMETRIC ALGORITHMS

There are two types of key-based encryption, symmetric (or secret-key) and asymmetric (or public-key) algorithms. Symmetric algorithms use the same key for encryption and decryption (or the decryption key is easily derived from the encryption key). According to Whitfield and Martin

(1976), Symmetric-key cryptography refers to encryption methods in which both the sender and receiver share the same key (or, less commonly, in which their keys are different, but related in an easily computable way). This was the only kind of encryption publicly known until June 1976. Asymmetric algorithms use a different key for encryption and decryption, and the decryption key cannot be derived from the encryption key.

Symmetric algorithms can be divided into stream ciphers and block ciphers. Stream ciphers can encrypt a single bit of plaintext at a time, whereas block ciphers take a number of bits (typically 64 bits in modern ciphers), and encrypt them as a single unit. An example of a symmetric algorithm is Digital Encryption Standard (DES). The Data Encryption Standard (DES) and the Advanced Encryption Standard (AES) are block cipher designs which have been designated cryptography standards by the US government.

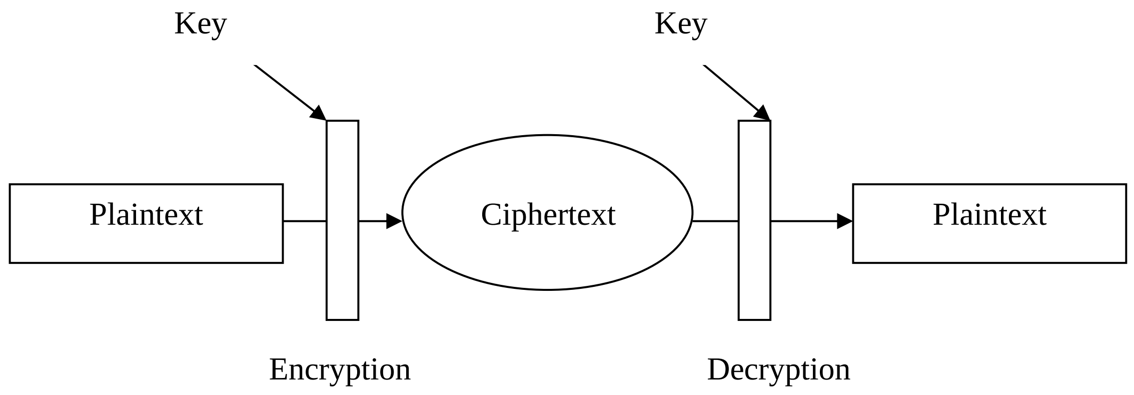


Figure 2.2: Symmetric Algorithms

Asymmetric ciphers (also called public-key cryptography) make a public key universally available, while only one individual possesses the private key. When data is encrypted with the public key, it can only be decrypted with the private key, and vice versa. Public key cryptography adds a very significant benefit - it can serve to authenticate a source (e.g. a digital signature).

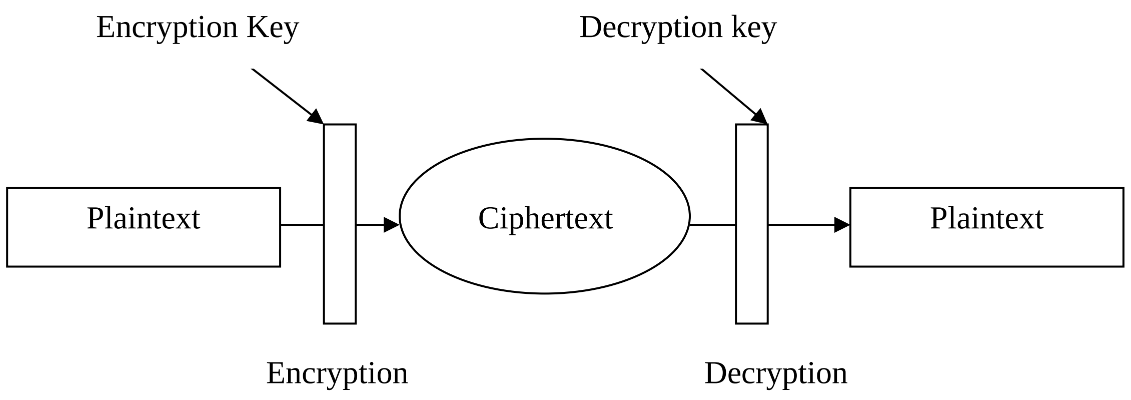


Figure 2.3: Asymmetric Algorithms

Public key cryptography was invented by Whitfield Diffie and Martin Hellman in 1975. An example of an asymmetric algorithm is RSA (x5.net, 2012). In general, symmetric algorithms execute much faster than asymmetric ones. In real applications, they are often used together, with a public-key algorithm encrypting a randomly generated encryption key, while the random key encrypts the actual message using a symmetric algorithm. This combination is commonly referred to as a digital envelope.

2.7 CIPHERS

A cipher is an encrypted text that is not readable to humans. The encrypted text is called a cipher. According to Haris (2014), there are two basic types of encryption ciphers:

i. substitution cipher and

ii. transposition cipher (permutation).

The **substitution cipher** replaces bits, characters, or blocks of characters with different bits, characters, or blocks. The **transposition cipher** does not replace the original text with different text, but moves the original text around. It rearranges the bits, characters, or blocks of characters to hide the original meaning.

2.7.1 Substitution Cipher

A substitution cipher uses a key to know how the substitution should be carried out. In the Caesar Cipher, each letter is replaced with the letter three places beyond it in the alphabet. This is referred to as a shift alphabet. If the Caesar Cipher is used with the English alphabet, when George wants to encrypt a message of "FBI," the encrypted message would be "IEL." Substitution is used in today's algorithms, but it is extremely complex compared to this example. Many different types of substitutions take place usually with more than one alphabet.

2.7.2 Transposition Cipher

In a **transposition cipher**, permutation is used, meaning that letters are scrambled. The key determines the positions that the characters are moved to, as illustrated in Figure 2.4. This is a simplistic example of a transposition cipher and only shows one way of performing transposition. According to CISSE, when introduced with complex mathematical functions, transpositions can become quite sophisticated and difficult to break. Most ciphers used today use long sequences of complicated substitutions and permutations together on messages. The key value is inputted into the algorithm and the result is the sequence of operations (substitutions and permutations) that are performed on the plaintext.

Simple substitution and transposition ciphers are vulnerable to attacks that perform **frequency analysis**. In every language, there are words and patterns that are used more often than others. For instance, in the English language, the words "the," "and," "that," and "is" are very frequent patterns of letters used in messages and conversation. The beginning of messages usually starts "Hello" or "Dear" and ends with "Sincerely" or "Goodbye." These patterns help attackers figure out the transformation between plaintext to ciphertext, which enables them to figure out the key that was used to perform the transformation. It is important for cryptosystems to not reveal these patterns.

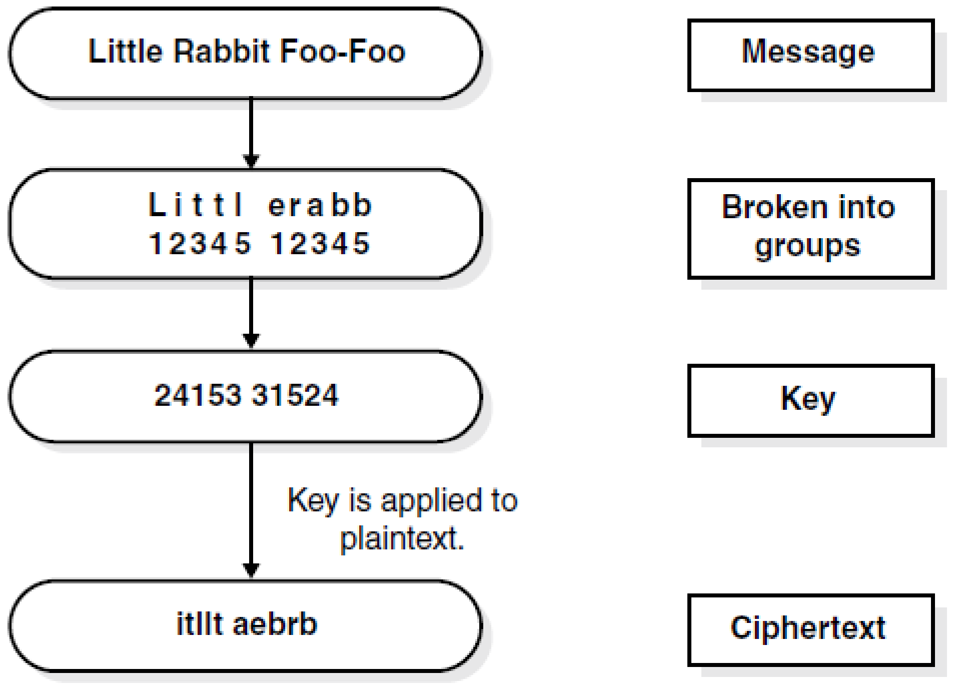


Fig 2.4: Transposition cipher

More complex algorithms usually use more than one alphabet for substitution and permutation, which reduces the vulnerability to frequency analysis. The more complicated the algorithm, the more the resulting text (ciphertext) differs from the plaintext; thus, the matching of these types of patterns becomes more difficult.

Running and Concealment Ciphers

More of the spy-novel-type ciphers would be the running key cipher and the concealment cipher. The ***running key cipher*** could use a key that does not require an electronic algorithm and bit alterations, but clever steps in the physical world around us. For instance, a key in this type of cipher could be a book page, line number, and word count. If I get a message from my super-secret spy buddy and the message reads “14967.29937.91158,” this could mean for me to look at the first book in our predetermined series of books, the 49th page, 6th line down the page, and the 7th word in that line. So I write down this word, which is “cat.” The second set of numbers start with 2, so I go to the 2nd book, 99th page, 3rd line down, and write down the 7th word on that line, which is “is”. The last word I get from the 9th book in our series, the 11th page, 5th row, and 8th word in that row, which is “dead.” So now I have come up with my important secret message, which is “cat is dead.” This means nothing to me and I need to look for a new spy buddy.

Running key ciphers can be used in different and more complex ways. Another type of spy novel cipher is the **concealment cipher**. If other super-secret spy buddy and I decide our key value is every third word, then when I get a message from him, I will pick out every third word and write it down. So if he sends me a message that reads, “The saying, ‘The time is right’ is not cow language, so is now a dead subject.” Because my key is every third word, I come up with “The right cow is dead.” This again means nothing to me and I am now turning in my decoder ring. No matter which type of cipher is used, the roles of the algorithm and key are the same, even if they are not mathematical equations. In the running key cipher, the algorithm states that encryption and decryption will take place by choosing characters out of a predefined set of books. The key indicates the book, page, line, and word within that line. In substitution cipher, the algorithm dictates that substitution will take place using a predefined alphabet or sequence of characters, and the key indicates that each character will be replaced with the third character that follows it in that sequence of characters.

In actual mathematical structures, the algorithm is a set of mathematical functions that will be performed on the message and the key can indicate in which order these functions take place.

So even if an attacker knows the algorithm, say the predefined set of books, if he does not know the key, the message is still useless to him.

2.8 TYPES OF CRYPTOGRAPHIC ALGORITHMS

There are several ways of classifying cryptographic algorithms (Dooley, 2013). For purposes of this project work, they will be categorized based on the number of keys that are employed for encryption and decryption, and further defined by their application and use. The three types of algorithms that will be reviewed for the purpose of this project work are:

i. Secret Key Cryptography (SKC): Uses a single key for both encryption and decryption
ii. Public Key Cryptography (PKC): Uses one key for encryption and another for decryption
iii. Hash Functions: Uses a mathematical transformation to irreversibly "encrypt" information.

2.8.1 Secret Key Cryptography

With secret key cryptography, a single key is used for both encryption and decryption. The sender uses the key (or some set of rules) to encrypt the plaintext and sends the ciphertext to the receiver. The receiver applies the same key (or ruleset) to decrypt the message and recover the plaintext. Because a single key is used for both functions, secret key cryptography is also called symmetric encryption. With this form of cryptography, it is obvious that the key must be known to both the sender and the receiver; that, in fact, is the secret. The biggest difficulty with this approach, of course, is the distribution of the key.

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

Stream ciphers come in several flavors but two are worth mentioning here. Self-synchronizing stream ciphers calculate each bit in the keystream as a function of the previous n bits in the keystream. It is termed "self-synchronizing" because the decryption process can stay synchronized with the encryption process merely by knowing how far into the n-bit keystream it is.

One problem is error propagation; a garbled bit in transmission will result in n garbled bits at the receiving side. Synchronous stream ciphers generate the keystream in a fashion independent of the message stream but by using the same keystream generation function at sender and receiver. While stream ciphers do not propagate transmission errors, they are, by their nature, periodic so that the keystream will eventually repeat.

i. Block ciphers can operate in one of several modes; the following four are the most important:
ii. Electronic Codebook (ECB) mode is the simplest, most obvious application: the secret key is used to encrypt the plaintext block to form a ciphertext block. Two identical plaintext blocks, then, will always generate the same ciphertext block. Although this is the most common mode of block ciphers, it is susceptible to a variety of brute-force attacks.

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

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

v. Output Feedback (OFB) mode is a block cipher implementation conceptually similar to a synchronous stream cipher. OFB prevents the same plaintext block from generating the same ciphertext block by using an internal feedback mechanism that is independent of both the plaintext and ciphertext bitstreams (http://www.i.thiyagaraaj.com).

2.8.2 Public-key cryptography

Symmetric-key cryptosystems use the same key for encryption and decryption of a message, though a message or group of messages may have a different key than others. A significant disadvantage of symmetric ciphers is the key management necessary to use them securely. Each distinct pair of communicating parties must, ideally, share a different key, and perhaps each ciphertext exchanged as well. The number of keys required increases as the square of the number of network members, which very quickly requires complex key management schemes to keep them all straight and secret. The difficulty of securely establishing a secret key between two communicating parties, when a secure channel does not already exist between them, also presents a chicken-and-egg problem which is a considerable practical obstacle for cryptography users in the real world. In a groundbreaking 1976 paper, Whitfield Diffie and Martin Hellman proposed the notion of *public-key* (also, more generally, called *asymmetric key*) cryptography in which two different but mathematically related keys are used—a *public* key and a *private* key (Whitfield et al, 1976).

2.9 CRYPTANALYSIS

There are several algorithms for cryptography. Therefore, the essence of cryptanalysis is to try to break any of the algorithms. For example, Peter Shor broke an encryption algorithm in 1995 using quantum computing technology to carry out the factorization that enabled him carry out the cryptanalysis. Cryptanalysts carry out cryptographic attacks which are designed to subvert the security of cryptographic algorithms, and they are used to attempt to decrypt data without prior access to a key. This is part of Cryptanalysis, which is the art of deciphering encrypted data. Cryptanalysis and Cryptography (the art of creating hidden writing, or ciphers) form the science of Cryptology.

2.10 Types of Cryptographic Attacks

Cryptographic attacks are designed to subvert the security of cryptographic algorithms, and they are used to attempt to decrypt data without prior access to a key. They are part of Cryptanalysis, which is the art of deciphering encrypted data. Cryptanalysis and Cryptography (the art of creating hidden writing, or ciphers) form the science of Cryptology (Jana, *et al.*, 2018).

2.10.1 Cryptographic Attack Methods

There are six related cryptographic attack methods, including three plaintext-based methods and three ciphertext-based methods:

Plain-text based Attacks	Known Plaintext	Chosen Plaintext	Adaptive Chosen Plaintext
Cipher-text based Attacks	Ciphertext only	Chosen Ciphertext	Adaptive Chosen Ciphertext

Fig 2.4: Cryptographic Attack methods

The methods above are used as the foundation of cryptographic attacks.

2.10.2 Known Plaintext and Ciphertext-Only Attacks

A known plaintext attack is an attack where a cryptanalyst has access to a plaintext and the corresponding ciphertext and seeks to discover a correlation between the two. A ciphertext-only attack is an attack where a cryptanalyst has access to a ciphertext but does not have access to corresponding plaintext. With simple ciphers, such as the Caesar Cipher, frequency analysis can be used to break the cipher.

2.10.3 Chosen Plaintext and Chosen Ciphertext Attacks

A chosen plaintext attack is an attack where a cryptanalyst can encrypt a plaintext of his choosing and study the resulting ciphertext. This is most common against asymmetric cryptography, where a cryptanalyst has access to a public key.

A chosen ciphertext attack is an attack where a cryptanalyst chooses a ciphertext and attempts to find a matching plaintext. This can be done with a decryption oracle (a machine that decrypts without exposing the key). This is also often performed on attacks versus public key encryption; it begins with a ciphertext and searches for matching publicly-posted plaintext data.

2.10.4 Adaptive Chosen Plaintext and Adaptive Chosen Ciphertext Attacks

In both adaptive attacks, a cryptanalyst chooses further plaintexts or ciphertexts (adapts the attack) based on prior results.

2.11 LINEAR CRYPTANALYSIS AND DIFFERENTIAL CRYPTANALYSIS

Differential cryptanalysis and linear cryptanalysis are related attacks used primarily against iterative symmetric key block ciphers. An iterative cipher (also called a product cipher) conducts multiple rounds of encryption using a subkey for each round. Examples include the Feistel Network used in DES and the State rounds used in AES. In both attacks, a cryptanalyst studies changes to the intermediate ciphertext between rounds of encryption. The attacks can be combined, which is called differential linear cryptanalysis (Blondeau and Gerard, 2011).

A goal of strong encryption is to produce ciphertexts that appear random where a small change in a plaintext results in a random change in the resulting ciphertext. This quality is called diffusion,5 and any changed ciphertext bit should have a 50% chance of being a 1 or a 0. Both attacks seek to discover non-randomness (cases where the 50% rule is broken) in an effort to discover potential subkeys.

2.11.1 Linear Cryptanalysis

Linear cryptanalysis is a known plaintext attack that requires access to large amounts of plaintext and ciphertext pairs encrypted with an unknown key (Heyes, 2002). It focuses on statistical analysis against one round of decryption on large amounts of ciphertext. The cryptanalyst decrypts each ciphertext using all possible subkeys for one round of encryption and studies the resulting intermediate ciphertext to seek the least random result. A subkey that produces the least random intermediate cipher for all ciphertexts becomes a candidate key (the most likely subkey).

2.11.2 Differential Cryptanalysis

Differential cryptanalysis is a chosen plaintext attack that seeks to discover a relationship between ciphertexts produced by two related plaintexts. It focuses on statistical analysis of two inputs and two outputs of a cryptographic algorithm.

A plaintext pair is created by applying a Boolean exclusive or (XOR) operation to a plaintext. For example, XOR the repeating binary string 10000000 to the plaintext. This creates a small difference (hence the term differential cryptanalysis) between the two. The cryptanalyst then encrypts the plaintext and its XORed pair using all possible subkeys, and it seeks signs of non-randomness in each intermediate ciphertext pair. The subkey that creates the least random pattern becomes the candidate key (Heys, 2002).

CHAPTER THREE

SYSTEM DESIGN

3.1 INTRODUCTION

In this chapter, the proposed framework for file security on personal computers using the Twofish block cipher algorithm is presented. The details entail the general architecture of a Twofish algorithm and the proposed framework for its application for ensuring confidentiality of personal file.

3.2 TWOFISH ALGORITHM

The Twofish cryptographic algorithm falls under the private key cryptography algorithm class in which the same key is used during both encryption and decryption phases. Twofish has a block size of 128 bits, and works with variable key length capped at 256 bits.

Twofish is a Feistel network which implies that: in each round, half of the data block is sent through an F function, and then XORed with the other half of the data block. Figure 3.2 shows the general architecture of the Twofish algorithm.

The input into the Fiestal network is a 128bits block of plain message to be encrypted. The basic process of encryption is as detailed in Figure 3.3. The decryption process also follows the basic steps itemized in figure 3.3 in which the input is the cipher text and the encryption key. However, the difference between encryption and decryption is that the key is reversed. In other words, if key $P_0,\ldots,P_n$ are used for encryption, the order of application during the decryption process will be $P_n,\ldots,P_0$.

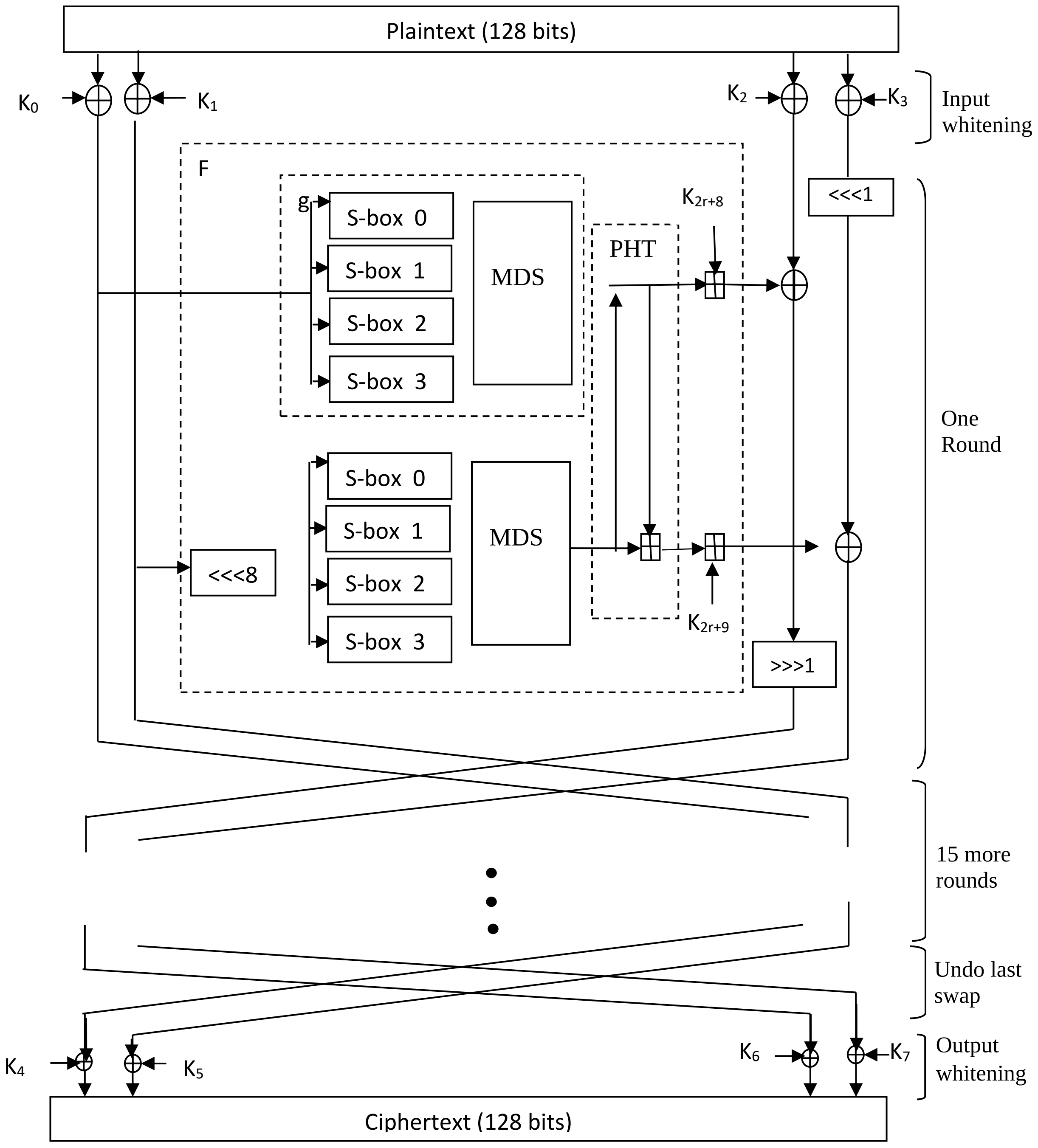


Figure 3.1: Twofish Algorithm Framework (Source: Schneier et al., 1998)

```
Twofish Algorithm

1. The 128bits plaintext block is split into four 32-bit words: $P_0,P_1,P_2,P_3$
2. XOR each $P_i$ with four Key words $K_i$ : $R_{0,i} = P_i \oplus K_i$ , $i = 0 \dots 3$
3. Repeat 16 times
    a. Partition first 32bits word into 4bytes and apply to S-box (Substitution)
    b. Rotate second 32bits word by 8bits, partition into 4bytes, and apply to
       S-box
    c. Diffuse each 32bits output from a and b by apply them on the MDS
       matrices
    d. Diffuse the two words from c by applying PHT (Pseudo-Hadamard
       Transform) and add two round key words to each
    e. XOR the third word with the first output from d, then rotate right by
       1bit
    f. Rotate left fourth word by 1bit and XOR it with the second output from
       d
    g. Exchange the two halves. That is, first and second words becomes third
       and fourth words, third and fourth words becomes first and second
       words.
4. XOR each 32bits with 4 round sub-keys to obtain the cipher text
```

Figure 3.2: Twofish Algorithm

3.3 PROPOSED ARCHITECTURE

This application proposed in this project relies on the Twofish encryption algorithm presented in Section 3.2. The proposed file encryption system is designed to be a standalone desktop application that allows user to upload a file, supply the encryption key, and then have the file encrypted on the physical storage. Figure 3.3 shows the detail of the proposed framework.

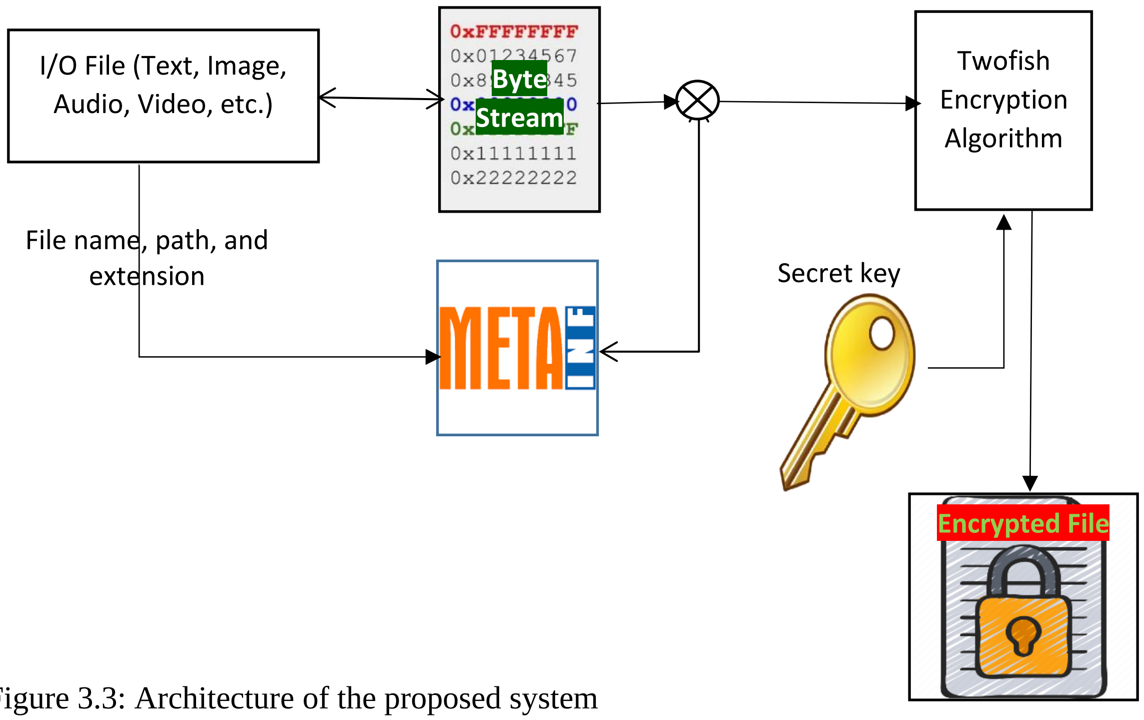

I/O File (Text, Image, Audio, Video, etc.)
0xFFFFFFFF
0x01234567
Byte
Stream
0x11111111
0x22222222
Twofish Encryption Algorithm
File name, path, and extension
META INF
Secret key
Encrypted File

Figure 3.3: Architecture of the proposed system

3.4 DATA FLOW DIAGRAM OF THE SYSTEM

The dataflow diagram is a modeling tool that allows us to picture a system as a network of functional processes, connected to one another by “pipelines” and “holding tanks” of data. The dataflow diagram is one of the most commonly used systems-modeling tools, particularly for operational systems in which the *functions* of the system are of paramount importance and more complex than the data that the system manipulates. DFDs were first used in the software engineering field as a notation for studying systems design issues.

3.3.1 DATA FLOW DIAGRAM LEVEL 0

DFD level 0 is the highest level view of the system, contains only one process which represents whole function of the system. It does not contain any data stores and the data is stored with in the process.

For constructing DFD level 0 diagram for the proposed approach we need two sources one is for source and another is for destination and a process.

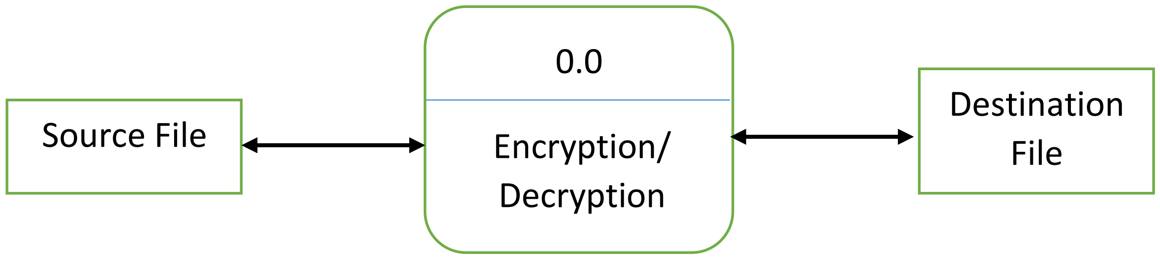


Figure 3.5: Data Flow Diagram Level 0

DFD level 0 is the basic data flow process, the main objective is to convert file into encrypted form and back to normal form.

3.3.2 DATA FLOW DIAGRAM LEVEL 1

For constructing DFD level 1, we need to identify and draw the process that make the level 0 process. In the project for transferring the personal data from source to destination, the personal data is first encrypted, transmitted or stored and later decrypted.

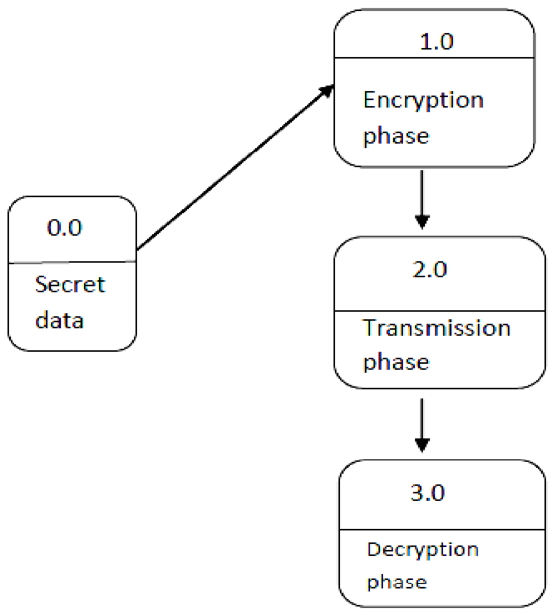


Figure 3.6: Data Flow Diagram Level 1

In this data flow diagram, the secret data is sent to the encryption phase for embedding the data into the image for generating the carrier image. In the next phase the carrier image is sent to the decryption phase through the transmission phase. The final phase is the decryption phase where the data is extracted from the image and displays the original message.

CHAPTER FOUR

SYSTEM IMPLEMENTATION

4.1 INTRODUCTION

This section explain various choices made while carrying out the implementation and the programming language employed. The section also contains the discussion on experiment as well as result obtained and analysis.

4.2 SYSTEM REQUIREMENTS

The system requirement entails the environmental settings that will ensure the smooth running of the application. These include the hardware and software requirements. The hardware requirements specify the set of minimum hardware configuration on which the application can perform optimally while the software requirements specify the set of software on which the application depends or set of libraries it relies on in order to work.

4.2.1 HARDWARE REQUIREMENTS

Although the underlying algorithm, Twofish, is fast on both 32-bit and 8-bit CPUs (smart cards, embedded chips, and the likes), and in hardware, a personal computer with the following configuration was used for the development and running of the application.

A Personal Computer with the following configuration

- 4GB RAM
- Intel Pentium Processor with speed of ~2.1GHz
- 4GB RAM

4.2.2 SOFTWARE REQUIREMENTS

For the application to run, the following software must be installed on the system.

- Java Runtime Environment (JRE version 1.6 or later)

- Operating System (e.g. Windows, Linux, Macintosh, etc.), since the application is platform independent.

4.3 CHOICE OF PROGRAMMING LANGUAGE

Java is the programming language of choice. The facts presented here are the key benefit of Java programming language.

4.3.1 Key Benefits of Java

Why use Java at all? Is it worth learning a new language and a new platform? This section explores some of the key benefits of Java.

i. *Write Once, Run Anywhere:* Sun identifies "Write once, run anywhere" as the core value proposition of the Java platform. Translated from business jargon, this means that the most important promise of Java technology is that you only have to write your application once--for the Java platform--and then you'll be able to run it *anywhere*.

Anywhere, that is, that supports the Java platform. Fortunately, Java support is becoming ubiquitous. It is integrated, or being integrated, into practically all major operating systems. It is built into the popular web browsers, which places it on virtually every Internet-connected PC in the world. It is even being built into consumer electronic devices, such as television set-top boxes, PDAs, and cell phones.

ii. *Security:* Another key benefit of Java is its security features. Both the language and the platform were designed from the ground up with security in mind. The Java platform allows users to download untrusted code over a network and run it in a secure environment in which it cannot do any harm: it cannot infect the host system with a virus, cannot read or write files from the hard drive, and so forth. This capability alone makes the Java platform unique.

The Java 2 Platform takes the security model a step further. It makes security levels and restrictions highly configurable and extends them beyond applets. As of Java 1.2, any Java code, whether it is an applet, a servlet, a JavaBeans component, or a complete Java application, can be run with restricted permissions that prevent it from doing harm to the host system.

The security features of the Java language and platform have been subjected to intense scrutiny by security experts around the world. Security-related bugs, some of them potentially serious, have been found and promptly fixed. Because of the security promises Java makes, it is big news when a new security bug is found. Remember, however, that no other mainstream platform can make security guarantees nearly as strong as those Java makes. If Java's security is not yet perfect, it has been proven strong enough for practical day-to-day use and is certainly better than any of the alternatives.

iii. *Network-centric Programming:* Sun's corporate motto has always been "The network is the computer." The designers of the Java platform believed in the importance of networking and designed the Java platform to be network-centric. From a programmer's point of view, Java makes it unbelievably easy to work with resources across a network and to create network-based applications using client/server or multitier architectures. This means that Java programmers have a serious head start in the emerging network economy.

iv. *Dynamic, Extensible Programs:* Java is both dynamic and extensible. Java code is organized in modular object-oriented units called *classes*. Classes are stored in separate files and are loaded into the Java interpreter only when needed. This means that an application can decide as it is running what classes it needs and can load them when it needs them. It also means that a program can dynamically extend itself by loading the classes it needs to expand its functionality.

The network-centric design of the Java platform means that a Java application can dynamically extend itself by loading new classes over a network. An application that takes advantage of these features ceases to be a monolithic block of code. Instead, it becomes an interacting collection of independent software components. Thus, Java enables a powerful new metaphor of application design and development.

v. *Internationalization:* The Java language and the Java platform were designed from the start with the rest of the world in mind. Java is the only commonly used programming language that has internationalization features at its very core, rather than tacked on as an afterthought. While most programming languages use 8-bit characters that represent only the alphabets of English and Western European languages, Java uses 16-bit Unicode characters that represent the phonetic alphabets and ideographic character sets of the entire world. Java's internationalization features are not restricted to just low-level character representation, however. The features permeate the Java platform, making it easier to write internationalized programs with Java than it is with any other environment.

vi. *Performance:* Java programs are compiled to a portable intermediate form known as byte codes, rather than to native machine-language instructions. The Java Virtual Machine runs a Java program by interpreting these portable byte-code instructions. This architecture means that Java programs are faster than programs or scripts written in purely interpreted languages, but they are typically slower than C and C++ programs compiled to native machine language. Keep in mind, however, that although Java programs are compiled to byte code, not all of the Java platform is implemented with interpreted byte codes. For efficiency, computationally intensive portions of the Java platform--such as the string-manipulation methods--are implemented using native machine code.

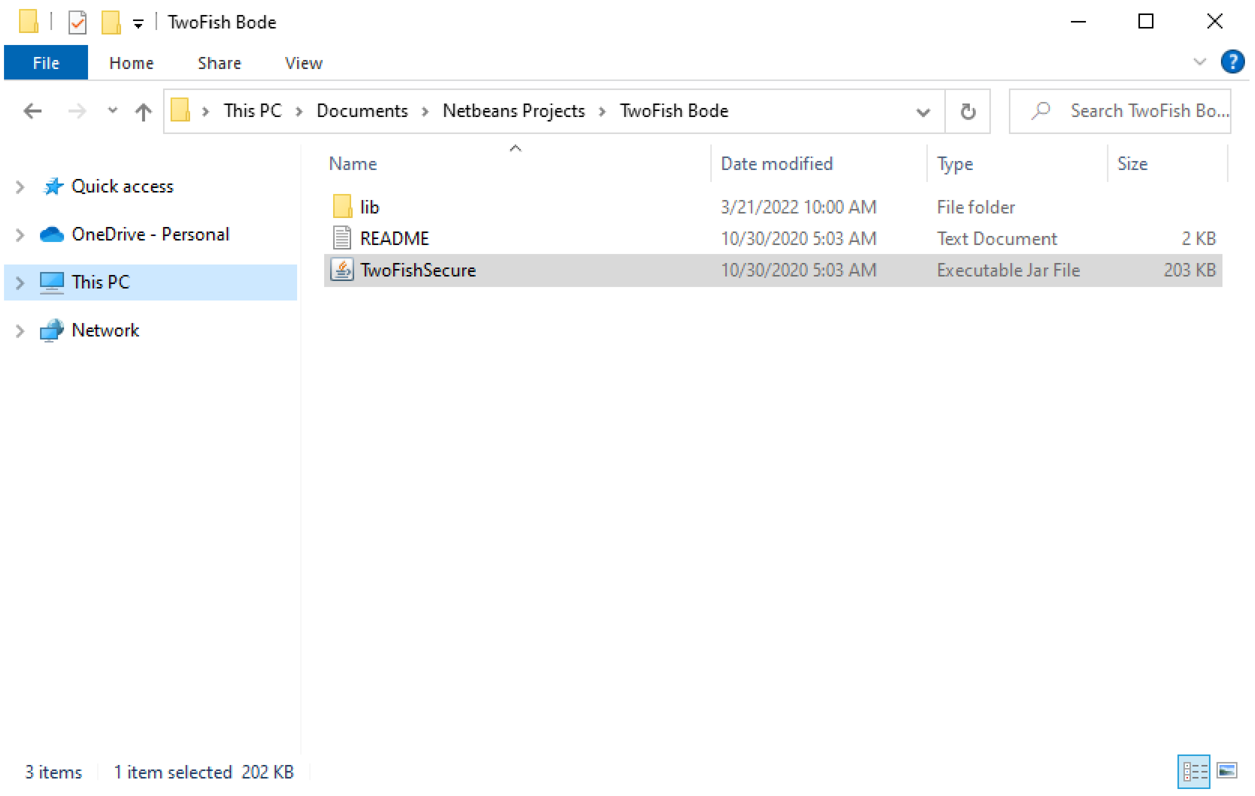
Although early releases of Java suffered from performance problems, the speed of the Java VM has improved dramatically with each new release. The VM has been highly tuned and optimized in many significant ways. Furthermore, many implementations include a just-in-time compiler, which converts Java byte codes to native machine instructions on the fly. Using sophisticated JIT compilers, Java programs can execute at speeds comparable to the speeds of native C and C++ applications.

Java is a portable, interpreted language; Java programs run almost as fast as native, non-portable C and C++ programs. Performance used to be an issue that made some programmers avoid using Java. Now, with the improvements made in Java 1.2, performance issues should no longer keep anyone away. In fact, the winning combination of performance plus portability is a unique feature no other language can offer.

vii. *Programmer Efficiency and Time-to-Market:* Java is an elegant language combined with a powerful and well-designed set of APIs. Programmers enjoy programming in Java and are usually amazed at how quickly they can get results with it. Studies have consistently shown that switching to Java increases programmer efficiency. Because Java is a simple and elegant language with a well-designed, intuitive set of APIs, programmers write better code with fewer bugs than for other platforms, again reducing development time.

4.4 DEPLOYMENT PROCEDURE

The bundled application is contained in a folder which can be distributed as a zipped file. The zipped folder when extracted contains three files, which include: a lib folder, which contains the libraries that the application depends on, a jar file, and a text document that contains guideline on the software usage, as shown in Figure 4.1.



Name	Date modified	Type	Size
lib	3/21/2022 10:00 AM	File folder	
README	10/30/2020 5:03 AM	Text Document	2 KB
TwoFishSecure	10/30/2020 5:03 AM	Executable Jar File	203 KB

Figure 4.1: Extracted Application Folder for File Security using Twofish Algorithm

The user is required to copy the extracted files in a folder in which the system user has a write permission. That is, the application will be allowed by the operating system to create temporary files. The application can then be launched by double-click the *jar* file. This loads the application splash screen as shown in Figure 4.2.

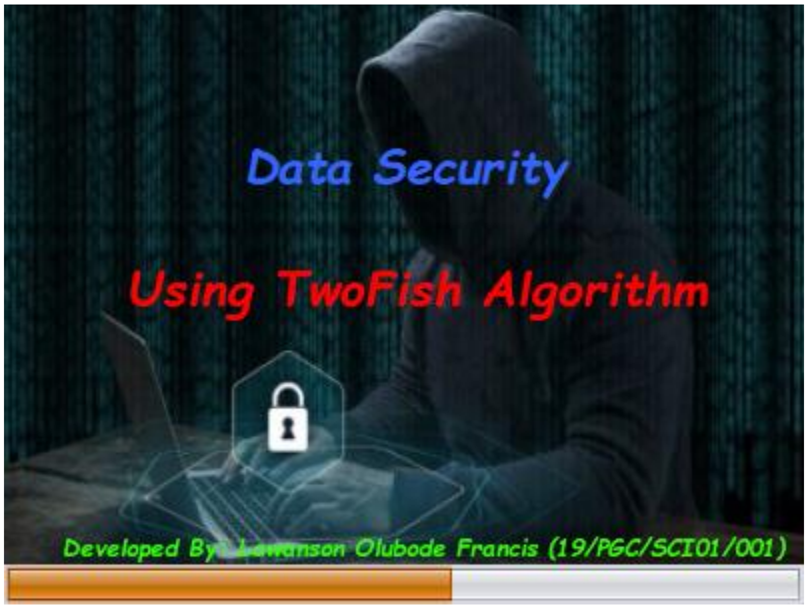


Figure 4.2: Application Splash Screen

The splash screen displays information about the application briefly and allows the necessary runtime modules to be loaded before displaying the application main window. The application main window, as shown in Figure 4.3 contains menu items that links to various pages of the application where the user can perform both encryption and decryption of files.

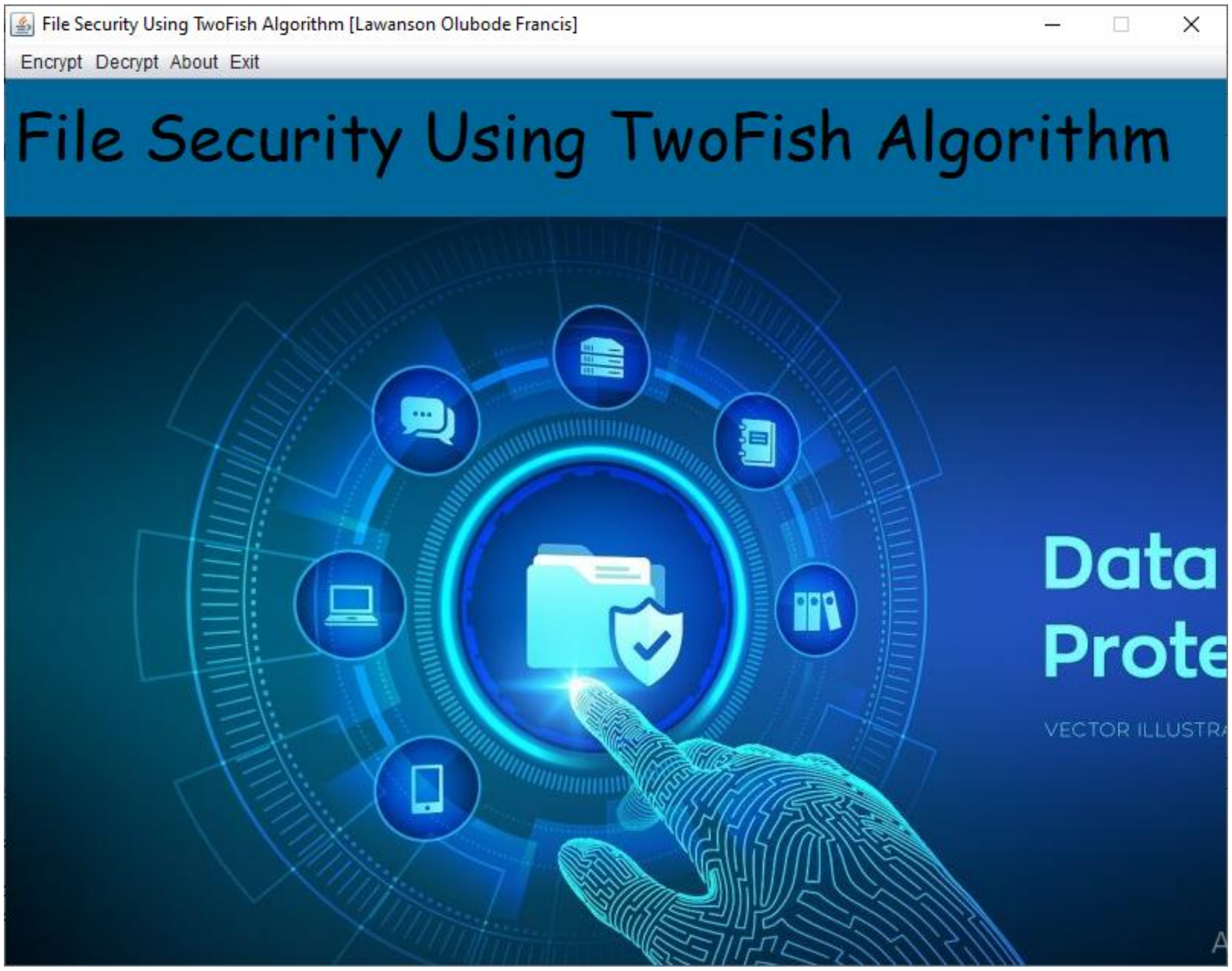
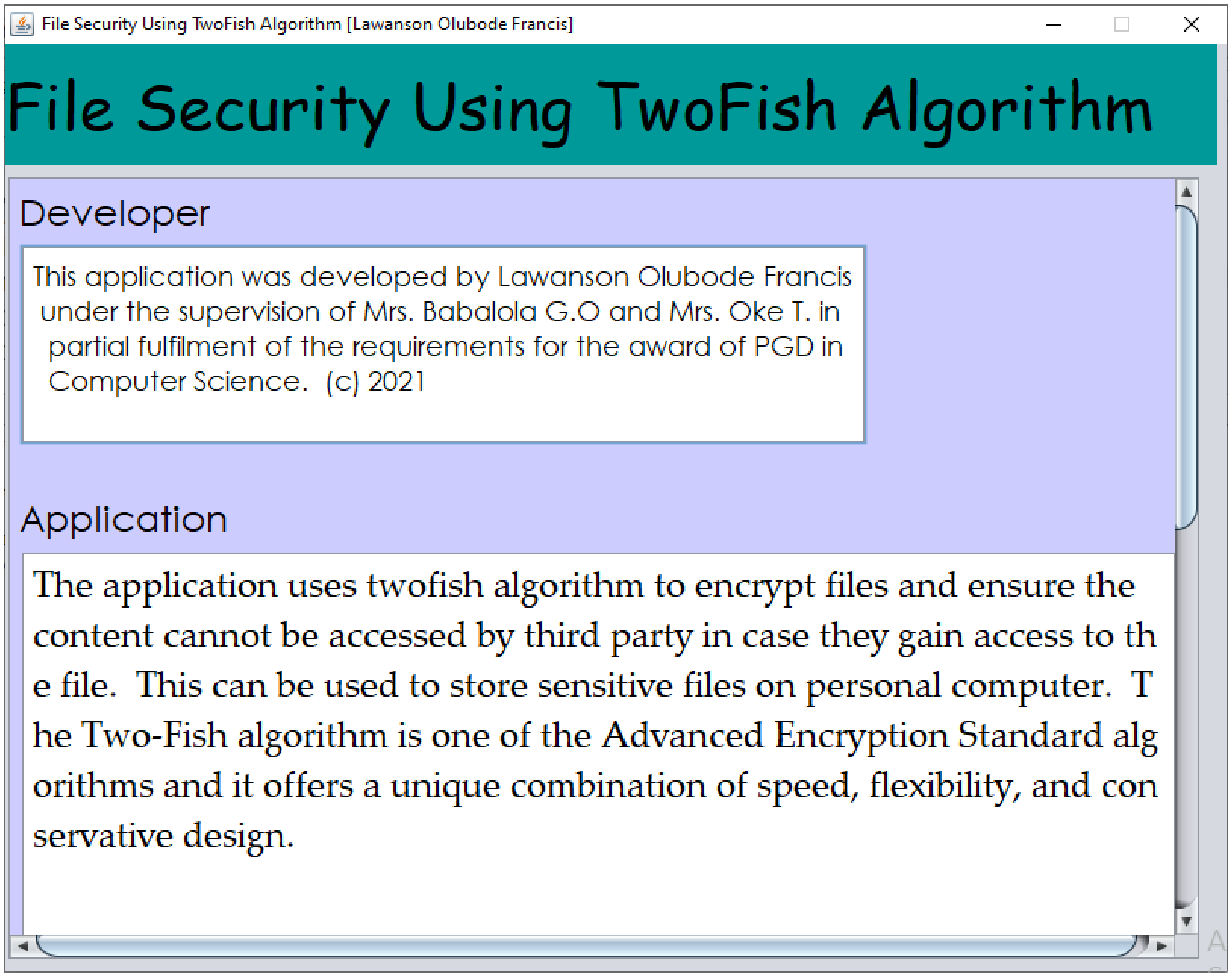

File Security Using TwoFish Algorithm [Lawanson Olubode Francis]
Encrypt Decrypt About Exit
File Security Using TwoFish Algorithm
Data
Prote
VECTOR ILLUSTR

Figure 4.3: Application Home Page

The *Exit* menu link terminates the application while the *About* menu link displays the About page. The about page contains brief description of the application internal working procedure as well as information about the application developer. Figure 4.4 shows the About page.


File Security Using TwoFish Algorithm [Lawanson Olubode Francis]

File Security Using TwoFish Algorithm

Developer

This application was developed by Lawanson Olubode Francis under the supervision of Mrs. Babalola G.O and Mrs. Oke T. in partial fulfilment of the requirements for the award of PGD in Computer Science. (c) 2021

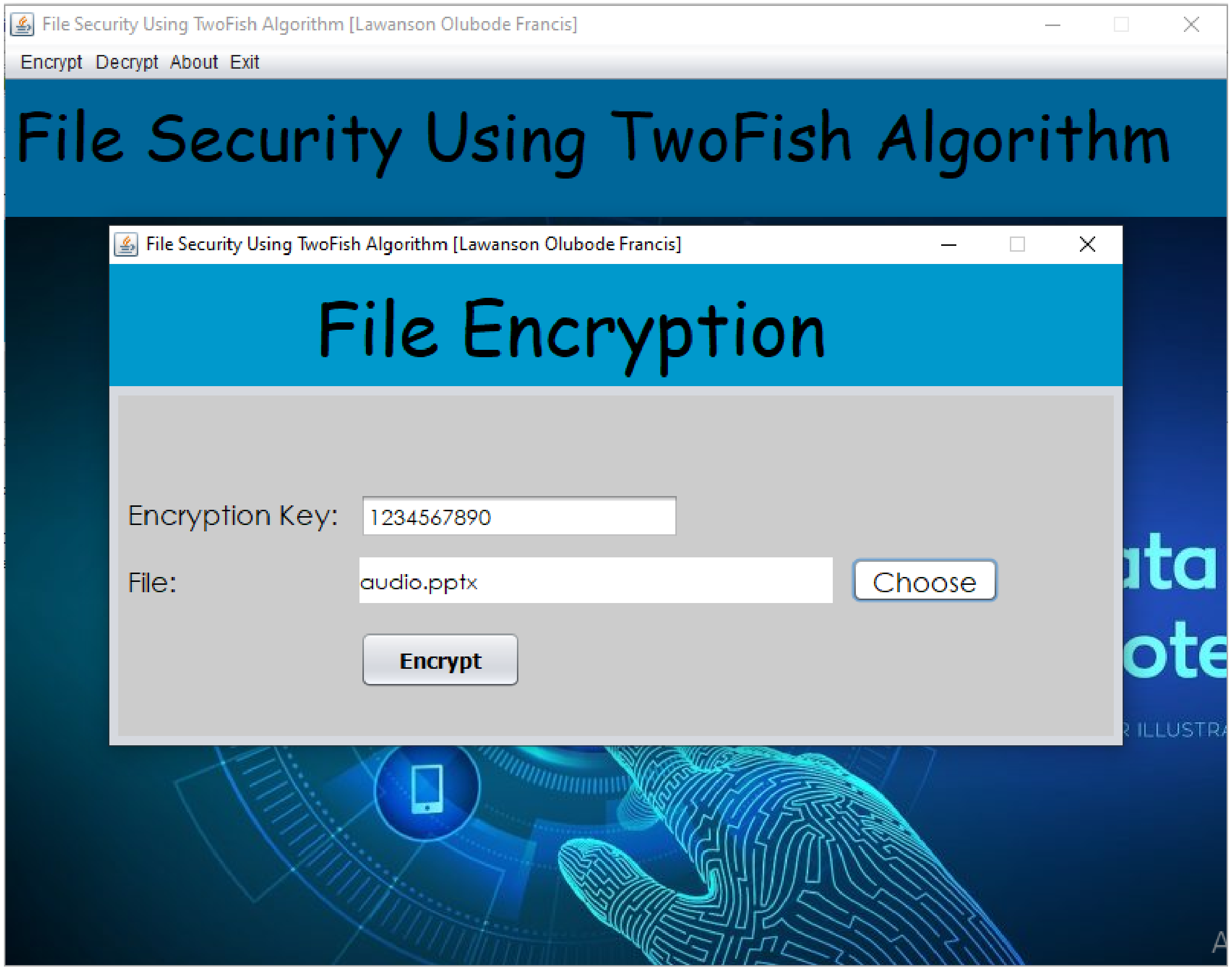
Application

The application uses twofish algorithm to encrypt files and ensure the content cannot be accessed by third party in case they gain access to th e file. This can be used to store sensitive files on personal computer. T he Two-Fish algorithm is one of the Advanced Encryption Standard alg orithms and it offers a unique combination of speed, flexibility, and con servative design.

Figure 4.4: About Page

File Encryption and Decryption

The encryption and decryption process are similar. Since TwoFish algorithm is a private key encryption algorithm, the same key used for encryption must be supplied during the decryption phase. To encrypt a file, the user supplies the Secret Key, selects the file to be encrypted and clicks on the Encrypt button as shown in Figure 4.5. The application reads the file into an array of bytes, extract the file extension from the file name, converts the file extension into bytes and append it to the array of bytes, encrypts the fusion of the file and its meta-data using the two fish encryption algorithm, writes the encrypted data into file and saves it with the extension *.file*. The original file is then deleted from the system. However, if the user desires to keep a copy of the original

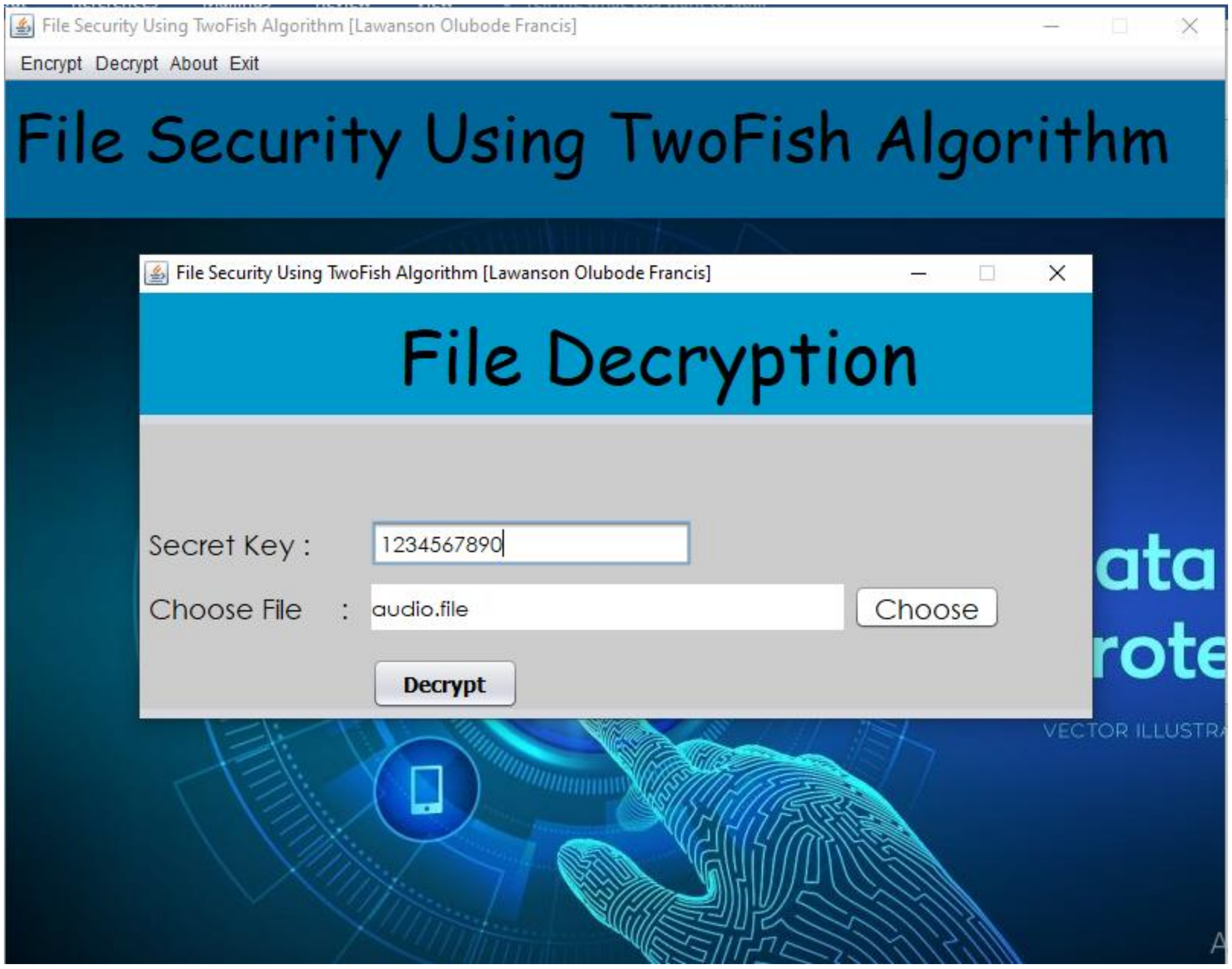

File Security Using TwoFish Algorithm [Lawanson Olubode Francis]
Encrypt Decrypt About Exit
File Security Using TwoFish Algorithm
File Security Using TwoFish Algorithm [Lawanson Olubode Francis]
File Encryption
Choose
Encrypt

Encryption Key: 1234567890

File: audio.pptx

Figure 4.5: File Encryption Interface

For the decryption process, the user clicks on the *Decrypt* menu link, which displays the decryption window.


File Security Using TwoFish Algorithm [Lawanson Olubode Francis]
Encrypt Decrypt About Exit
File Security Using TwoFish Algorithm
File Security Using TwoFish Algorithm [Lawanson Olubode Francis]
File Decryption
Secret Key :
Choose File :
Choose
Decrypt

Secret Key : 1234567890

Choose File : audio.file

Figure 4.6: File Decryption Interface

The user is expected to supply the secret key and select the corresponding encrypted file. The application attempts to decrypt the selected file using supplied key. If the key is correct, the file extension is retrieved from the extracted data bytes. The decrypted file is then written to the same folder where the encrypted file is with the correct file extension.

CHAPTER FIVE

CONCLUSION AND RECOMMENDATION

5.1 CONCLUSION

With increasing adoption of information and communication technologies across various sectors, the issue of privacy and personal security becomes highly important. Most system users are unaware of the potential dangers associated with the use of internet and hence often fall prey to attackers and cyber criminals. Personal observation of the researcher showed that most system user rely on password as their best defense mechanism against most forms of cyber-attack. However, it is a common knowledge that passwords often find its use in access control – granting authorized user privilege to resources. In scenarios where the attacker gain access to the system through password authentication bypass or guessing, or gained physical access to system due to negligence on the part of the system owner, there is no mechanism to check the user’s activity on the system. Cryptography aim to render information useless to unauthorized third party. In this project, a desktop-based application for file encryption and decryption using Twofish algorithms was proposed and developed. The choice of the algorithm is due to its open source, high speed of encryption and decryption, key flexibility, and low hardware resource consumption. The application can be used on personal computers to encrypt and decrypt sensitive files. Two layers of security is provided by the system since users must be authenticated before accessing the application interface, and also provide the cipher key, which could be unique for each file.

5.2 RECOMMENDATION

In order to protect users from threats and secure personal files on computer systems, it is recommended that the application be adopted for personal use.

5.3 FUTURE WORK

In order to guarantee optimal security, it is advisable that the user should use different keys for each file to be encrypted. This will lead to the use of large number of keys which might be impossible to remember without writing them down, which is also not a good approach to password/key management. Hence, there is need to design and integrate a key generation and management scheme into the application. This will improve the overall security of the application.

REFERENCES

Andy, W.(1998). Tips and tricks: XOR Encryption “http://www.andyw.com/director/Xor.asp.

Anjana, D., & Ramya, B.(2017). Twofish algorithm implementation for lab to provide data security with predictive analysis. International journal of Research in Engineering and Technology. eISSN: 2395-0056 ; pISSN:2395-0072.

Blondeau, C., & Gérard, B. (2011). Multiple differential cryptanalysis: theory and practice. In International Workshop on Fast Software Encryption (pp. 35-54). Springer, Berlin, Heidelberg.

Dooley, J. F. (2013). A brief history of cryptology and cryptographic algorithms (pp. 1-9). New York: Springer.

Duncan, B., & Whittington, M. (2016). Cloud cyber-security: Empowering the audit trail. International Journal on Advances in Security Volume 9, Number 3 & 4, 2016.

Gajendra, S., Vinod, S. & Uday, p. (2016). Different image encryption techniques-survey and overview. International Journal of Advanced Research in Computer Science and Software Engineering. ISSN:2277 128x

Harris, S, (2014). CISSP Cryptography training: components, protocols and authentication. Retrieved from https://www.techtarget.com/searchsecurity/feature/CISSP-cryptography-training-Components-protocols-and-authentication#:~:text=Some%20of%20the%20symmetric%20algorithms,Advanced%20Encryption%20Standard%20(AES) on 16/02/2021.

Heys, H. M. (2002). A tutorial on linear and differential cryptanalysis. Cryptologia, 26(3), 189-221.

Jana, B., Chakraborty, M., Mandal, T., & Kule, M. (2018, May). An Overview on Security Issues in Modern Cryptographic Techniques. In Proceedings of 3rd International Conference on Internet of Things and Connected Technologies (ICIoTCT) (pp. 26-27).

Krishan, G. (2013). Different image encryption and decryption techniques and KA image cryptography. International Journal of Advanced Computational Engineering and Networking, ISSN:2320-2106.

Much, A., Budi, P. & Alamsyah.(2016). Implementation Twofish algorithm for data security in a communication network using library chilkat encryption active. Journal of Theoretical and Applied Information Technology. ISSN: 1992-8645 & E-ISSN 1817-3195.

Omar, F., Falah, Y. & Subhi, R (2017). A survey and analysis of the image encryption methods. International Journal of Applied Engineering Research.

Pushpa, B. (2020). Hybrid data encryption algorithm for secure medical data transmission in cloud environment. In 2020 Fourth International Conference on Computing Methodologies and Communication (ICCMC) (pp. 329-334). IEEE.

Qadir, S., & Quadri, S. M. K. (2016). Information availability: An insight into the most important attribute of information security. Journal of Information Security, 7(3), 185-194.

Ranjan, K.,Fathimath, S.,Ganesh, A. & Surendra, S.(2017). A survey on key(s) and keyless image encryption techniques. Cybertics and Information Technologies. Vol 17. ISSN:1311-9702; online ISSN:1314-4081.

Shaza, D.,Ahmed,K. &Saife-Eldin F.(2015). A performance comparison of encryption algorithms AES and DES. International Journal of Engineering Research & Technology. ISSN:2278-0181.

Shi, L., John, T. & Changliang,G. (2013). A review of optical image encryption techniques. Journal of Electrical, Electronic and Communication Engineering and Architecture. Optics &Laser Technology 57 (2014)327-342.

William, S., Lawrie, B., Mick, B. & Micheal H. (2012). Computer security principles and practice. ISBN-13: 978-0-13-277506-9

Yaqoob, I., Ahmed, E., ur Rehman, M. H., Ahmed, A. I. A., Al-garadi, M. A., Imran, M., & Guizani, M. (2017). The rise of ransomware and emerging security challenges in the Internet of Things. Computer Networks, 129, 444-458.

Yin, Y., Stecke, K. E., & Li, D. (2018). The evolution of production systems from Industry 2.0 through Industry 4.0. International Journal of Production Research, 56(1-2), 848-861.

APPENDIX

SPLASH SCREEN

```
package bode.files;

import java.awt.Color;
import javax.swing.UIManager;
import javax.swing.plaf.ColorUIResource;

public class Screen extends javax.swing.JFrame {

   public Screen() {
      initComponents();
   }

   @SuppressWarnings("unchecked")
   // <editor-fold defaultstate="collapsed" desc="Generated Code">
   private void initComponents() {

      jPanel1 = new javax.swing.JPanel();
      jLabel2 = new javax.swing.JLabel();
      jLabel4 = new javax.swing.JLabel();
      jLabel5 = new javax.swing.JLabel();
      jProgressBar1 = new javax.swing.JProgressBar();
      jLabel1 = new javax.swing.JLabel();
      jLabel6 = new javax.swing.JLabel();

      setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
      setCursor(new java.awt.Cursor(java.awt.Cursor.DEFAULT_CURSOR));
      setFocusCycleRoot(false);
      setUndecorated(true);
      setResizable(false);

      jPanel1.setCursor(new java.awt.Cursor(java.awt.Cursor.WAIT_CURSOR));

      jLabel2.setBackground(new java.awt.Color(255, 255, 255));
      jLabel2.setFont(new java.awt.Font("Comic Sans MS", 0, 12)); // NOI18N
      jLabel2.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
      jLabel2.setText("Loading...");

      jLabel4.setFont(new java.awt.Font("Comic Sans MS", 3, 24)); // NOI18N
      jLabel4.setForeground(new java.awt.Color(51, 102, 255));
      jLabel4.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
      jLabel4.setText("Data Security");

      jLabel5.setFont(new java.awt.Font("Comic Sans MS", 3, 12)); // NOI18N
```

```
        jLabel5.setForeground(new java.awt.Color(51, 255, 0));
        jLabel5.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
        jLabel5.setText("Developed By: Lawanson Olubode Francis (19/PGC/SCI01/001)  ");

        jProgressBar1.setBackground(new java.awt.Color(0, 0, 0));
        jProgressBar1.setFont(new java.awt.Font("Comic Sans MS", 0, 12)); // NOI18N
        jProgressBar1.setForeground(new java.awt.Color(255, 255, 255));

        jLabel1.setIcon(new javax.swing.ImageIcon(getClass().getResource("/secure/files/Background
Image.jpg"))); // NOI18N
        jLabel1.setText("jLabel1");

        jLabel6.setFont(new java.awt.Font("Comic Sans MS", 3, 24)); // NOI18N
        jLabel6.setForeground(new java.awt.Color(255, 0, 0));
        jLabel6.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
        jLabel6.setText("Using TwoFish Algorithm");

        javax.swing.GroupLayout jPanel1Layout = new javax.swing.GroupLayout(jPanel1);
        jPanel1.setLayout(jPanel1Layout);
        jPanel1Layout.setHorizontalGroup(
            jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(jLabel5, javax.swing.GroupLayout.PREFERRED_SIZE, 400,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(jLabel4, javax.swing.GroupLayout.PREFERRED_SIZE, 400,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(jProgressBar1, javax.swing.GroupLayout.PREFERRED_SIZE, 400,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addGap(160, 160, 160)
                .addComponent(jLabel2, javax.swing.GroupLayout.PREFERRED_SIZE, 70,
javax.swing.GroupLayout.PREFERRED_SIZE))
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jLabel6, javax.swing.GroupLayout.PREFERRED_SIZE, 400,
javax.swing.GroupLayout.PREFERRED_SIZE))
            .addComponent(jLabel1, javax.swing.GroupLayout.PREFERRED_SIZE, 400,
javax.swing.GroupLayout.PREFERRED_SIZE)
        );
        jPanel1Layout.setVerticalGroup(
            jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING,
false)
                    .addGroup(jPanel1Layout.createSequentialGroup()
                        .addGap(50, 50, 50)
                        .addComponent(jLabel4, javax.swing.GroupLayout.PREFERRED_SIZE, 60,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                    .addComponent(jLabel6, javax.swing.GroupLayout.PREFERRED_SIZE, 50,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                    .addComponent(jLabel5))
                .addComponent(jLabel1, javax.swing.GroupLayout.PREFERRED_SIZE, 280,
javax.swing.GroupLayout.PREFERRED_SIZE))
            .addGap(0, 0, 0)
            .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addComponent(jProgressBar1, javax.swing.GroupLayout.PREFERRED_SIZE, 20,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addComponent(jLabel2, javax.swing.GroupLayout.PREFERRED_SIZE, 20,
javax.swing.GroupLayout.PREFERRED_SIZE)))
        );

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(jPanel1, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(jPanel1, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
        );

        setSize(new java.awt.Dimension(400, 300));
        setLocationRelativeTo(null);
    }// </editor-fold>

    /**
     * @param args the command line arguments
     */


    // Variables declaration - do not modify
    private javax.swing.JLabel jLabel1;
    public javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JPanel jPanel1;
    public javax.swing.JProgressBar jProgressBar1;
    // End of variables declaration
}
```

ENCRYPTION CODE

```
package bode.files;

import java.io.File;
import java.io.ByteArrayOutputStream;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import javax.swing.JFileChooser;
import javax.swing.JOptionPane;

/**
 *
 * @author Developer
 */
public class Encrypt extends javax.swing.JFrame {

    /**
     * Creates new form EncryptImage
     */
    public Encrypt() {
        setTitle("Secure Files");
        initComponents();
    }

    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        jPanel1 = new javax.swing.JPanel();
        jLabel2 = new javax.swing.JLabel();
        jPanel2 = new javax.swing.JPanel();
        jLabel4 = new javax.swing.JLabel();
        jTextField1 = new javax.swing.JTextField();
        jLabel7 = new javax.swing.JLabel();
        jButton1 = new javax.swing.JButton();
        file_name = new javax.swing.JLabel();
        jButton2 = new javax.swing.JButton();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setResizable(false);
```

```
jPanel1.setBackground(new java.awt.Color(0, 153, 204));
jPanel1.setPreferredSize(new java.awt.Dimension(800, 87));

jLabel2.setFont(new java.awt.Font("Comic Sans MS", 0, 48)); // NOI18N
jLabel2.setText("File Encryption");

javax.swing.GroupLayout jPanel1Layout = new javax.swing.GroupLayout(jPanel1);
jPanel1.setLayout(jPanel1Layout);
jPanel1Layout.setHorizontalGroup(
   jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
   .addGroup(jPanel1Layout.createSequentialGroup()
      .addGap(135, 135, 135)
      .addComponent(jLabel2)
      .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
);
jPanel1Layout.setVerticalGroup(
   jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
   .addComponent(jLabel2, javax.swing.GroupLayout.DEFAULT_SIZE, 80, Short.MAX_VALUE)
);

jPanel2.setBackground(new java.awt.Color(204, 204, 204));

jLabel4.setFont(new java.awt.Font("Century Gothic", 0, 18)); // NOI18N
jLabel4.setText("Encryption Key:");

jTextField1.setFont(new java.awt.Font("Century Gothic", 0, 14)); // NOI18N
jTextField1.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(java.awt.event.ActionEvent evt) {
      jTextField1ActionPerformed(evt);
   }
});

jLabel7.setFont(new java.awt.Font("Century Gothic", 0, 18)); // NOI18N
jLabel7.setText("File: ");

jButton1.setBackground(new java.awt.Color(255, 255, 255));
jButton1.setFont(new java.awt.Font("Century Gothic", 0, 18)); // NOI18N
jButton1.setText("Choose");
jButton1.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(java.awt.event.ActionEvent evt) {
      jButton1ActionPerformed(evt);
   }
});

file_name.setBackground(new java.awt.Color(255, 255, 255));
file_name.setFont(new java.awt.Font("Century Gothic", 0, 14)); // NOI18N
file_name.setOpaque(true);
```

```
        jButton2.setFont(new java.awt.Font("Tahoma", 1, 14)); // NOI18N
        jButton2.setText("Encrypt");
        jButton2.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseClicked(java.awt.event.MouseEvent evt) {
                jButton2MouseClicked(evt);
            }
        });

        javax.swing.GroupLayout jPanel2Layout = new javax.swing.GroupLayout(jPanel2);
        jPanel2.setLayout(jPanel2Layout);
        jPanel2Layout.setHorizontalGroup(
            jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel2Layout.createSequentialGroup()
                .addContainerGap()
                .addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addComponent(jLabel4, javax.swing.GroupLayout.PREFERRED_SIZE, 140,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jLabel7))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                .addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addComponent(jButton2, javax.swing.GroupLayout.PREFERRED_SIZE, 106,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addGroup(jPanel2Layout.createSequentialGroup()
                        .addComponent(file_name, javax.swing.GroupLayout.PREFERRED_SIZE, 310,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                        .addComponent(jButton1))
                    .addComponent(jTextField1, javax.swing.GroupLayout.PREFERRED_SIZE, 210,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addContainerGap(63, Short.MAX_VALUE))
        );
        jPanel2Layout.setVerticalGroup(
            jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel2Layout.createSequentialGroup()
                .addGap(62, 62, 62)
                .addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(jLabel4, javax.swing.GroupLayout.PREFERRED_SIZE, 30,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jTextField1, javax.swing.GroupLayout.PREFERRED_SIZE, 30,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                .addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING,
false)
                    .addComponent(jLabel7, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                    .addComponent(jButton1, javax.swing.GroupLayout.PREFERRED_SIZE, 30,
javax.swing.GroupLayout.PREFERRED_SIZE)
```

```
                .addComponent(file_name, javax.swing.GroupLayout.DEFAULT_SIZE, 
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
              .addGap(18, 18, 18)
              .addComponent(jButton2, javax.swing.GroupLayout.PREFERRED_SIZE, 38, 
javax.swing.GroupLayout.PREFERRED_SIZE)
              .addContainerGap(31, Short.MAX_VALUE))
        );

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
          layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
          .addComponent(jPanel1, javax.swing.GroupLayout.DEFAULT_SIZE, 664, Short.MAX_VALUE)
          .addGroup(layout.createSequentialGroup()
              .addContainerGap()
              .addComponent(jPanel2, javax.swing.GroupLayout.DEFAULT_SIZE, 
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
              .addContainerGap())
        );
        layout.setVerticalGroup(
          layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
          .addGroup(layout.createSequentialGroup()
              .addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED_SIZE, 80, 
javax.swing.GroupLayout.PREFERRED_SIZE)
              .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
              .addComponent(jPanel2, javax.swing.GroupLayout.PREFERRED_SIZE, 
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
              .addContainerGap())
        );

        pack();
        setLocationRelativeTo(null);
    }// </editor-fold>

    private void jTextField1ActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        JFileChooser fc=new JFileChooser();
        fc.showOpenDialog(null);

        File f=fc.getSelectedFile();
        fileName=f.getName();
        file_name.setText(fileName);
        fileParentPath=f.getParent();
        fileAbsolutePath=f.getAbsolutePath();
    }

    private void jButton2MouseClicked(java.awt.event.MouseEvent evt) {
        // TODO add your handling code here:
        EncryptFile();
    }

    private void EncryptFile()
    {
        try
        {
            String text=jTextField1.getText();
            long key=Long.parseLong(text);

            try
            {
                FileInputStream fis=new FileInputStream(fileAbsolutePath);
                try {
//////////////////////////////////////////////////////////////////////////////////////////
                    StringBuilder ext = new StringBuilder(fileName);                    //
                    ext = getFileExtension(ext);    //calling user defined function       //
//                    System.out.println("After All Proccess Extension: "+ext);           //
//////////////////////////////////////////////////////////////////////////////////////////
                    byte[] data = new byte[(fis.available() + ext.length())];

                    //l.setVisible(true);
                    //l.setTitle("Proceesing...Please wait...");

                    fis.read(data);

////////////////////////////////////////////////////////////////////////////////
                    data = addExtensionToFile(data, ext);                 //
////////////////////////////////////////////////////////////////////////////////

                    TwoFish twofish = new TwoFish();
                    data = twofish.encrypt(data,key);
/////////////////////////////////////////////////////////////////////////////////////
                    String encFileName = fileName;                          //
                    encFileName = encFileName.replaceAll(ext.toString(), ".enc");  //
//
/////////////////////////////////////////////////////////////////////////////////////
                    String fileOutputName=fileParentPath + "\\" + encFileName;
                    FileOutputStream fos = new FileOutputStream(fileOutputName);
                    fos.write(data);
                    fos.close();
                    fis.close();

/////////////////////////////////////////////////////////
```

```
            if(fileAbsolutePath.compareTo(fileOutputName)!=0){
              File f=new File(fileAbsolutePath);     //
              f.delete();                        //
            }
/////////////////////////////////////////////////////
              //l.setVisible(false);
              JOptionPane.showMessageDialog(null, "File Encrypted Successfully.");

            } catch (Exception e) {
              JOptionPane.showMessageDialog(null,"Some error accour while processing !!!");
            }
            jTextField1.setText(null);
            file_name.setText(null);
          }
          catch (Exception e)
          {
            JOptionPane.showMessageDialog(null,"Please Select File");
          }
        }
        catch (Exception e)
        {
          JOptionPane.showMessageDialog(null,"Invalid Key");
        }
    }
    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {

        try {
          for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {
            if ("Nimbus".equals(info.getName())) {
              javax.swing.UIManager.setLookAndFeel(info.getClassName());
              break;
            }
          }
        } catch (ClassNotFoundException | InstantiationException | IllegalAccessException |
javax.swing.UnsupportedLookAndFeelException ex) {
          java.util.logging.Logger.getLogger(Encrypt.class.getName()).log(java.util.logging.Level.SEVERE,
null, ex);
        }
        //</editor-fold>
        //</editor-fold>

        //</editor-fold>

        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
          @Override
          public void run() {
            new Encrypt().setVisible(true);
          }
        });
    }

    private String fileParentPath;
    private String fileName;
    private String fileAbsolutePath;
    // Variables declaration - do not modify
    private javax.swing.JLabel file_name;
    private javax.swing.JButton jButton1;
    private javax.swing.JButton jButton2;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JPanel jPanel2;
    private javax.swing.JTextField jTextField1;
    // End of variables declaration

    private StringBuilder getFileExtension(StringBuilder ext) throws Exception {

        ext.reverse();
        String strFileNname=ext.toString();
        char[] arrFileName=strFileNname.toCharArray();
        strFileNname="";

        int i=0;

        for(char c:arrFileName){
          if(c=='.')
            strFileNname=ext.substring(0, i);
          i++;
        }
        //System.out.println("After  "+str_fn);
        ext.delete(0, ext.length());
        ext.append(strFileNname).append(".");
        ext.reverse();

        return ext;
    }


    //Method to append extension into byte-data
    private byte[] addExtensionToFile(byte[] data,StringBuilder ext) throws Exception {
        byte appendExtension[]=ext.toString().getBytes();

        ByteArrayOutputStream bas=new ByteArrayOutputStream();
        bas.write(data);
        bas.write(appendExtension);
        data=bas.toByteArray();
        bas.close();

        return data;
    }
}
```

DECRYPTION CODE

```
package bode.files;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.RandomAccessFile;
import javax.swing.JFileChooser;
import javax.swing.JOptionPane;

/**
 *
 * @author Developer
 */
public class Decrypt extends javax.swing.JFrame {

    /**
     * Creates new form DecryptImage
     */
    public Decrypt() {
        initComponents();
    }

    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        jPanel1 = new javax.swing.JPanel();
        jLabel2 = new javax.swing.JLabel();
        jPanel2 = new javax.swing.JPanel();
        jTextField1 = new javax.swing.JTextField();
        jLabel7 = new javax.swing.JLabel();
        jButton1 = new javax.swing.JButton();
        jLabel6 = new javax.swing.JLabel();
        file_name = new javax.swing.JLabel();
        jButton2 = new javax.swing.JButton();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setResizable(false);

        jPanel1.setBackground(new java.awt.Color(0, 153, 204));
        jPanel1.setPreferredSize(new java.awt.Dimension(800, 87));
```

```
jLabel2.setFont(new java.awt.Font("Comic Sans MS", 0, 48)); // NOI18N
jLabel2.setText("File Decryption");

javax.swing.GroupLayout jPanel1Layout = new javax.swing.GroupLayout(jPanel1);
jPanel1.setLayout(jPanel1Layout);
jPanel1Layout.setHorizontalGroup(
   jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
   .addGroup(jPanel1Layout.createSequentialGroup()
      .addGap(170, 170, 170)
      .addComponent(jLabel2)
      .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
);
jPanel1Layout.setVerticalGroup(
   jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
   .addComponent(jLabel2, javax.swing.GroupLayout.DEFAULT_SIZE, 80, Short.MAX_VALUE)
);

jPanel2.setBackground(new java.awt.Color(204, 204, 204));

jTextField1.setFont(new java.awt.Font("Century Gothic", 0, 14)); // NOI18N
jTextField1.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(java.awt.event.ActionEvent evt) {
      jTextField1ActionPerformed(evt);
   }
});

jLabel7.setFont(new java.awt.Font("Century Gothic", 0, 18)); // NOI18N
jLabel7.setText("Choose File     : ");

jButton1.setBackground(new java.awt.Color(255, 255, 255));
jButton1.setFont(new java.awt.Font("Century Gothic", 0, 18)); // NOI18N
jButton1.setText("Choose");
jButton1.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(java.awt.event.ActionEvent evt) {
      jButton1ActionPerformed(evt);
   }
});

jLabel6.setFont(new java.awt.Font("Century Gothic", 0, 18)); // NOI18N
jLabel6.setText("Secret Key :");

file_name.setBackground(new java.awt.Color(255, 255, 255));
file_name.setFont(new java.awt.Font("Century Gothic", 0, 14)); // NOI18N
file_name.setHorizontalAlignment(javax.swing.SwingConstants.LEFT);
file_name.setText("audio.file");
file_name.setOpaque(true);
```

```
        jButton2.setFont(new java.awt.Font("Tahoma", 1, 14)); // NOI18N
        jButton2.setText("Decrypt");
        jButton2.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton2ActionPerformed(evt);
            }
        });

        javax.swing.GroupLayout jPanel2Layout = new javax.swing.GroupLayout(jPanel2);
        jPanel2.setLayout(jPanel2Layout);
        jPanel2Layout.setHorizontalGroup(
            jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel2Layout.createSequentialGroup()
                .addContainerGap()
                .addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addComponent(jLabel6, javax.swing.GroupLayout.PREFERRED_SIZE, 140,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jLabel7, javax.swing.GroupLayout.PREFERRED_SIZE, 140,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addComponent(jTextField1, javax.swing.GroupLayout.PREFERRED_SIZE, 210,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addGroup(jPanel2Layout.createSequentialGroup()
                        .addComponent(file_name, javax.swing.GroupLayout.PREFERRED_SIZE, 310,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(jButton1))
                    .addComponent(jButton2, javax.swing.GroupLayout.PREFERRED_SIZE, 97,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addContainerGap(54, Short.MAX_VALUE))
        );
        jPanel2Layout.setVerticalGroup(
            jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel2Layout.createSequentialGroup()
                .addContainerGap(63, Short.MAX_VALUE)
                .addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
                    .addComponent(jLabel6, javax.swing.GroupLayout.PREFERRED_SIZE, 30,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jTextField1, javax.swing.GroupLayout.PREFERRED_SIZE, 30,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                .addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING,
false)
                    .addComponent(jLabel7, javax.swing.GroupLayout.PREFERRED_SIZE, 30,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addGroup(jPanel2Layout.createSequentialGroup()
.addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addComponent(file_name, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                    .addGroup(jPanel2Layout.createSequentialGroup()
                        .addComponent(jButton1, javax.swing.GroupLayout.PREFERRED_SIZE, 30,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addGap(0, 0, Short.MAX_VALUE)))
                .addGap(18, 18, 18)
                .addComponent(jButton2, javax.swing.GroupLayout.PREFERRED_SIZE, 34,
javax.swing.GroupLayout.PREFERRED_SIZE))))
        );

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(jPanel2, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
            .addComponent(jPanel1, javax.swing.GroupLayout.DEFAULT_SIZE, 625, Short.MAX_VALUE)
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED_SIZE, 80,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(jPanel2, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap())
        );

        pack();
        setLocationRelativeTo(null);
    }// </editor-fold>

    private void jTextField1ActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:

    }

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        JFileChooser fc=new JFileChooser();
        fc.showOpenDialog(null);

        File f=fc.getSelectedFile();
        fileName=f.getName();
        file_name.setText(fileName);
        fileParentPath=f.getParent();
        fileAbsolutePath=f.getAbsolutePath();
//        System.out.println("Asbo: "+f.getAbsolutePath());
//        System.out.println("Parent: "+f.getParent());
//        System.out.println("Name: "+f.getName());
//        System.out.println("Path: "+f.getPath());

    }

    private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        DecryptFile();
    }

    private void DecryptFile(){
        try
        {
            String text=jTextField1.getText();
            long key=Long.parseLong(text);

            try
            {
                FileInputStream fis=new FileInputStream(fileAbsolutePath);
                try {
                    byte[]data=new byte[fis.available()];

                    //l.setVisible(true);
                    //l.setTitle("Proceesing...Please wait...");

                    fis.read(data);
                    TwoFish twofish = new TwoFish();
                    data = twofish.decrypt(data,key);
////////////////////////////////////////////////////////////////////////////////////////////
                    StringBuilder ext = new StringBuilder();                   //
                    ext = getFileExtensionFromFile(data);   //calling user defined function //
//                    System.out.println("Extension From File:"+ext);             //
                                                                          //
                    String decFileName=fileName;                                //
                    decFileName=decFileName.replaceFirst(".enc",ext.toString());      //
////////////////////////////////////////////////////////////////////////////////////////////

                    String fileOutputName=fileParentPath + "\\" + decFileName;

                    FileOutputStream fos=new FileOutputStream(fileOutputName);
                    fos.write(data);
////////////////////////////////////////////////////////////////////////////////////////////
                    data=removeExtensionFromFile(data, (short) ext.length(),decFileName);   //
////////////////////////////////////////////////////////////////////////////////////////
                fos.close();
                fis.close();

////////////////////////////////////////////////////////////////////
             if(fileAbsolutePath.compareTo(fileOutputName)!=0){     //
                File f=new File(fileAbsolutePath);                //
                f.delete();                                     //
             }                                                //
////////////////////////////////////////////////////////////////////

                //l.setVisible(false);
                JOptionPane.showMessageDialog(null,"File Decrypted Successfully.");

             } catch (Exception e) {
             }

             jTextField1.setText(null);
             file_name.setText(null);
          }
          catch (Exception e)
          {
          JOptionPane.showMessageDialog(null,"Please Select File");
          }
       }
       catch (Exception e)
       {
          JOptionPane.showMessageDialog(null,"Invalid Key");
       }
    }
    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
       /* Set the Nimbus look and feel */
       //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
       /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
        * For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
        */
       try {
          for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {
             if ("Nimbus".equals(info.getName())) {
                javax.swing.UIManager.setLookAndFeel(info.getClassName());
                break;
             }
          }
       } catch (ClassNotFoundException ex) {
          java.util.logging.Logger.getLogger(Decrypt.class.getName()).log(java.util.logging.Level.SEVERE,
null, ex);
        } catch (InstantiationException ex) {
          java.util.logging.Logger.getLogger(Decrypt.class.getName()).log(java.util.logging.Level.SEVERE,
null, ex);
        } catch (IllegalAccessException ex) {
          java.util.logging.Logger.getLogger(Decrypt.class.getName()).log(java.util.logging.Level.SEVERE,
null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {
          java.util.logging.Logger.getLogger(Decrypt.class.getName()).log(java.util.logging.Level.SEVERE,
null, ex);
        }
        //</editor-fold>
        //</editor-fold>

        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
          public void run() {
            new Decrypt().setVisible(true);
          }
        });
    }

    private String fileParentPath;
    private String fileName;
    private String fileAbsolutePath;
    //private String filename;
    // Variables declaration - do not modify
    private javax.swing.JLabel file_name;
    private javax.swing.JButton jButton1;
    private javax.swing.JButton jButton2;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JPanel jPanel2;
    private javax.swing.JTextField jTextField1;
    // End of variables declaration
    private StringBuilder getFileExtensionFromFile(byte[] data) throws Exception {

        StringBuilder ext=new StringBuilder("");
        for(int i=data.length-1;;i--){

//
if(data[i]>=65&&data[i]<=90||data[i]>=97&&data[i]<=122||data[i]==46||data[i]>=48&&data[i]<=57)
          if(Character.isLetterOrDigit(data[i])||data[i]==46)
            ext.insert(0, (char)data[i]);
          else
```

```
                break;
            }
            return ext;
        }

        private byte[] removeExtensionFromFile (byte[] data,short cnt,String decFileName) throws Exception {
                cnt *= 2;
                RandomAccessFile raf = new RandomAccessFile(fileParentPath + "\\" + decFileName, "rwd");
                raf.seek(raf.length()-1);

                for (int i = 0; i < cnt; i++)
                    raf.setLength(raf.length()-1);
                raf.close();
            return data;
        }
}
```

TWO FISH ALGORITHM

```
import bode.Galua256;

/**
 * @author developer
 */
public class TwoFish {
    private static final byte[][] RS = new byte[][] {
            new byte[] { (byte)0x01, (byte)0xA4, (byte)0x55, (byte)0x87, (byte)0x5A, (byte)0x58, (byte)0xDB,
(byte)0x9E},
            new byte[] { (byte)0xA4, (byte)0x56, (byte)0x82, (byte)0xF3, (byte)0x1E, (byte)0xC6, (byte)0x68,
(byte)0xE5},
            new byte[] { (byte)0x02, (byte)0xA1, (byte)0xFC, (byte)0xC1, (byte)0x47, (byte)0xAE, (byte)0x3D,
(byte)0x19},
            new byte[] { (byte)0xA4, (byte)0x55, (byte)0x87, (byte)0x5A, (byte)0x58, (byte)0xDB, (byte)0x9E,
(byte)0x03}
    };
    private static final byte[][] MDS = new byte[][] {
            new byte[] { (byte)0x01, (byte)0xEF, (byte)0x5B, (byte)0x5B},
            new byte[] { (byte)0x5B, (byte)0xEF, (byte)0xEF, (byte)0x01},
            new byte[] { (byte)0xEF, (byte)0x5B, (byte)0x01, (byte)0xEF},
            new byte[] { (byte)0xEF, (byte)0x01, (byte)0xEF, (byte)0x5B},
    };
    private static final byte[] t00 = { 0x8, 0x1, 0x7, 0xD, 0x6, 0xF, 0x3, 0x2, 0x0, 0xB, 0x5, 0x9, 0xE, 0xC,
0xA, 0x4};
    private static final byte[] t01 = { 0xE, 0xC, 0xB, 0x8, 0x1, 0x2, 0x3, 0x5, 0xF, 0x4, 0xA, 0x6, 0x7, 0x0,
0x9, 0xD};
    private static final byte[] t02 = { 0xB, 0xA, 0x5, 0xE, 0x6, 0xD, 0x9, 0x0, 0xC, 0x8, 0xF, 0x3, 0x2, 0x4,
0x7, 0x1};
    private static final byte[] t03 = { 0xD, 0x7, 0xF, 0x4, 0x1, 0x2, 0x6, 0xE, 0x9, 0xB, 0x3, 0x0, 0x8, 0x5,
0xC, 0xA};
```

```
   //
   private static final byte[] t10 = { 0x2, 0x8, 0xB, 0xD, 0xF, 0x7, 0x6, 0xE, 0x3, 0x1, 0x9, 0x4, 0x0, 0xA,
0xC, 0x5};
   private static final byte[] t11 = { 0x1, 0xE, 0x2, 0xB, 0x4, 0xC, 0x3, 0x7, 0x6, 0xD, 0xA, 0x5, 0xF, 0x9,
0x0, 0x8};
   private static final byte[] t12 = { 0x4, 0xC, 0x7, 0x5, 0x1, 0x6, 0x9, 0xA, 0x0, 0xE, 0xD, 0x8, 0x2, 0xB,
0x3, 0xF};
   private static final byte[] t13 = { 0xB, 0x9, 0x5, 0x1, 0xC, 0x3, 0xD, 0xE, 0x6, 0x4, 0x7, 0xF, 0x2, 0x0,
0x8, 0xA};

   static int[] encrypt(int[] plainText, int[] key) {
      return encrypt(plainText, key, false);
   }

   static int[] encrypt(int[] plainText, int[] key, boolean debug) {
      final int[] roundKey01 = roundKeys(key, 0);
      final int[] roundKey23 = roundKeys(key, 1);
      final int[] roundKey45 = roundKeys(key, 2);
      final int[] roundKey67 = roundKeys(key, 3);
      // whitening
      int[] whitened = whitening(plainText, roundKey01[0], roundKey01[1], roundKey23[0],
roundKey23[1]);
      //
      if(debug) {
         System.out.println("whitened:");
         //Utils.printInternal(whitened);
      }
      //
      for(int i = 0; i < 16; i++) {

         whitened = encryptionRound(whitened, key, i);
         if(debug) {
            System.out.println("R"+i + ":");
            if(i % 2 == 0) {
               //Utils.printInternal(whitened);
            }
         }
         whitened = new int[] {whitened[2], whitened[3], whitened[0], whitened[1]};
         if(debug && i % 2 != 0) {
            //Utils.printInternal(whitened);
         }
      }
      // Swapping
      whitened = new int[] {whitened[2], whitened[3], whitened[0], whitened[1]};
      whitened = whitening(whitened, roundKey45[0], roundKey45[1], roundKey67[0], roundKey67[1]);
      return whitened;
   }
```

```
    public static int[] decrypt(int[] cypheredText, int[] key) {
        return decrypt(cypheredText, key, false);
    }

    public static int[] decrypt(int[] cypheredText, int[] key, boolean debug) {
        if(debug) {
            System.out.println("Cyphered text:");
            //Utils.printInput(cypheredText);
        }
        final int[] roundKey01 = roundKeys(key, 0);
        final int[] roundKey23 = roundKeys(key, 1);
        final int[] roundKey45 = roundKeys(key, 2);
        final int[] roundKey67 = roundKeys(key, 3);
        // whitening
        int[] whitened = whitening(cypheredText, roundKey45[0], roundKey45[1], roundKey67[0],
roundKey67[1]);
        if(debug) {
            System.out.println("Whitened:");
            //Utils.printInternal(whitened);
        }
        //
        whitened = new int[] {whitened[2], whitened[3], whitened[0], whitened[1]};
        for(int i = 15; i >= 0; i--) {
            whitened = decryptionRound(whitened, key, i);
            if(debug) {
                System.out.println("R"+ (i + 1) + ":");
                if(i % 2 == 0) {
                    //Utils.printInternal(whitened);
                }
            }
            whitened = new int[] {whitened[2], whitened[3], whitened[0], whitened[1]};
            if(debug && i % 2 != 0) {
                //Utils.printInternal(whitened);
            }
        }
        whitened = whitening(whitened, roundKey01[0], roundKey01[1], roundKey23[0], roundKey23[1]);
        if(debug) {
        System.out.println("Whitened:");
        //Utils.printInternal(whitened);
        }
        return whitened;

    }

    public static int[] whitening(int[] plainText, int k0, int k1, int k2, int k3) {
        return new int[] {
                plainText[0] ^ k0,
                plainText[1] ^ k1,
                plainText[2] ^ k2,
                plainText[3] ^ k3
        };
    }

    public static int[] encryptionRound(int[] input, int[] key, int round) {
        final int[] s = getS(key);
        int t0 = h(input[0],                        s[1], s[0]);
        int t1 = h(Integer.rotateLeft(input[1], 8), s[1], s[0]);
        int[] pPht = pht(t0, t1);
        final int[] roundKeys2r_8_2r_9 = roundKeys(key, round + 4);
        //
        final int f0 = pPht[0] + roundKeys2r_8_2r_9[0];
        final int f1 = pPht[1] + roundKeys2r_8_2r_9[1];
        //
        int c2 = Integer.rotateRight((f0 ^ input[2]), 1);
        int c3 = (f1 ^ Integer.rotateLeft(input[3], 1));
        //
        return new int[] { input[0], input[1], c2, c3 };
    }

    public static int[] decryptionRound(int[] input, int[] key, int round) {
        final int[] s = getS(key);
        int t0 = h(input[2],                        s[1], s[0]);
        int t1 = h(Integer.rotateLeft(input[3], 8), s[1], s[0]);
        final int[] pPht = pht(t0, t1);
        final int[] roundKeys = roundKeys(key, round + 4);
        //
        final int f0 = pPht[0] + roundKeys[0];
        final int f1 = pPht[1] + roundKeys[1];
        //
        final int p2 = Integer.rotateLeft(input[0], 1) ^ f0;
        final int p3 = Integer.rotateRight(input[1] ^ f1, 1);
        //
        return new int[] {  p2, p3, input[2], input[3]};

    }

    public static int[] pht(int a, int b) {
        int a1 = a + b;
        int b1 = (a + 2 * b);
        return new int[] {a1, b1};
    }

    public static int h(int input, int l0, int l1) {
        Galua256 galua256 = new Galua256((byte)0b01101001);
        final byte[] x = asBytes(input);
        final byte[] y = asBytes(l1);
        final byte[] z = asBytes(l0);
        final byte[] input11 = new byte[] {
            q1((byte) (q0((byte) (q0(x[0]) ^ y[0])) ^ z[0])),
            q0((byte) (q0((byte) (q1(x[1]) ^ y[1])) ^ z[1])),
            q1((byte) (q1((byte) (q0(x[2]) ^ y[2])) ^ z[2])),
            q0((byte) (q1((byte) (q1(x[3]) ^ y[3])) ^ z[3])),
        };
        return fromBytes(multiply(galua256, MDS, input11));
    }

    public static byte q0(byte input) {
        byte a0 = (byte)((input >> 4) & 0xF);
        byte b0 = (byte)(input & 0xF);
        byte a1 = (byte)(a0 ^ b0);
        byte b1 = (byte)(a0 ^ ((b0 & 1) << 3 | b0 >> 1) ^ ((8*a0) & 0xF));
        byte a2 = t00[a1];
        byte b2 = t01[b1] ;
        byte a3 = (byte)(a2 ^ b2);
        byte b3 = (byte)(a2 ^ ((b2 & 1) << 3 | b2 >> 1) ^ ((8*a2) & 0xF));
        byte a4 = t02[a3];
        byte b4 = t03[b3];
        return (byte)((b4 << 4) | a4);
    }

    public static byte q1(byte input) {
        byte a0 = (byte)((input >> 4) & 0xF);
        byte b0 = (byte)(input & 0xF);
        byte a1 = (byte)(a0 ^ b0);
        byte b1 = (byte)(a0 ^ ((b0 & 1) << 3 | b0 >> 1) ^ ((8*a0) & 0xF));
        byte a2 = t10[a1];
        byte b2 = t11[b1];
        byte a3 = (byte)(a2 ^ b2);
        byte b3 = (byte)(a2 ^ ((b2 & 1) << 3 | b2 >> 1) ^ ((8*a2) & 0xF));
        byte a4 = t12[a3];
        byte b4 = t13[b3];
        return (byte)((b4 << 4) | a4);
    }

    public static int[] getS(int[] key) {
        final int m0 = key[0];
        final int m1 = key[1];
        final int m2 = key[2];
        final int m3 = key[3];
        final int S0 = RS(m0, m1);
        final int S1 = RS(m2, m3);
        return new int[] { S0, S1};
    }
```

```
private static int RS(int X, int Y) {
   byte[] x = asBytes(X);
   byte[] y = asBytes(Y);
   byte[] XY = new byte[8];
   // Merging x and y
   System.arraycopy(x, 0, XY, 0, 4);
   System.arraycopy(y, 0, XY, 4, 4);
   //
   final byte[][] matrix = RS;
   Galua256 galua = new Galua256((byte)0b01001101);
   //
   byte[] S = multiply(galua, matrix, XY);
   return fromBytes(S);
}

private static byte[] multiply(Galua256 galua, byte[][] matrix, byte[] vector) {
   byte[] S = new byte[vector.length];
   for(int i = 0; i < matrix.length; i++) {
      final byte[] RSrow = matrix[i];
      S[i] = galua.multiply(RSrow[0], vector[0]);
      for(int j = 1; j < RSrow.length; j++) {
         S[i] = galua.add(S[i], galua.multiply(RSrow[j], vector[j]));
      }
   }
   return S;
}

public static int[] roundKeys(int[] key, int round) {
   final int m0 = key[0];
   final int m1 = key[1];
   final int m2 = key[2];
   final int m3 = key[3];
   //
   final int[] Me = new int[] { m0, m2};
   final int[] Mo = new int[] { m1, m3};
   //
   final int rho = (1 << 24) | (1 << 16) | (1 << 8) | 1;
   final int Ai = h(2 * round * rho, Me[0], Me[1]);
   final int Bi = Integer.rotateLeft(h((2 * round + 1) * rho, Mo[0], Mo[1]), 8);
   final int[] pPht = pht(Ai, Bi);
   final int K2i = pPht[0];
   final int K2i_1 = Integer.rotateLeft(pPht[1], 9);
   return new int[] { K2i, K2i_1};
}

public static byte[] asBytes(int intValue) {
   return new byte[] {
         (byte)(intValue),
                (byte)(intValue >>> 8),
                (byte)(intValue >>> 16),
                (byte)(intValue >>> 24),
        };
    }

    public static int fromBytes(byte[] bytes) {
        int S0 = 0;
        for(int i = 0; i < 4; i++) {
            S0 |= ((0xFF & bytes[i]) << (i * 8));
        }
        return S0;
    }
}
```