**ABSTRACT**

In this project we are focusing on creating a cryptographic algorithm that enhances security. This will help people to transfer information over the air without the fear of data loss or manipulation of data.

Cryptography is one of the major concerned areas of computer and data security. The proposed algorithm is an advanced key-dependent version of the AES algorithm originally given by the NIST. In the proposed design the S-Box is key dependent. The algorithm involves key expansion, together with S-Box rotation and this property can be used to make the S-Box key-dependent, hence providing a better security to the block cipher. With the introduction of the encrypted S-Box , it serves well in security critical applications such as voting system, banking system, etc.

**ACKNOWLEDGEMENT**

The satisfaction that accompanies the successful completion of any task would be incomplete without the mention of people whose ceaseless cooperation made it possible, whose constant guidance and encouragement helped us throughout.

We sincerely express deep sense of gratitude to our guide **Mr. Sankusu Sharma ,** and Head of Department **Mrs. Neeta Patil (Computer Engineering),** for their valuable guidance, inspiration and constructive suggestions that helped us in the preparation of this project.

We would also like to take the opportunity to thank our professors for their creative inputs and support. All of the aforementioned people have a big part in clearing our doubts about the project and giving us confidence to go about for the project.

**Gladston Eric D’souza**

**Kevin Pinto**

**Shivshankar Suresh**

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **Sr. No** | **Contents** | **Page No** |
| 1.0 | Abstract | 1 |
| 2.0 | Project Overview | 2 |
| 3.0 | Introduction and Motivation   1. Aim and Objectives 2. Literature Surveyed 3. Existing System 4. Scope of the Project 5. Motivation | 4  5  6  12  12  13 |
| 4.0 | Problem Statement | 14 |
| 5.0 | Requirement Analysis | 17 |
| 6.0 | Project design:   1. Functional Requirements: (DFD) 2. Procedural Design 3. Data Dictionary /Data Structures: 4. Entity Relationship Diagram: 5. Relational Table 6. Uml Diagrams | 23  23  24  29  30  31  38 |
| 7.0 | Implementation Details | 49 |
| 8.0 | Technology Used | 63 |
| 9.0 | Test Cases | 64 |
| 10.0 | Project Time Line | 65 |
| 11.0 | Task Distribution | 69 |
| 12.0 | Conclusion & Future Work | 70 |
| 13.0 | References | 71 |
| 14.0 | Appendix | 72 |

List of Figures

List of Tables

# CHAPTER NO. I : INTRODUCTION

1. The purpose of this document is to study the behavior of the proposed cryptographic algorithm named ***“KES(Key Enabled S-Box) Algorithm”.*** This system provides a comparative study into the performance of the existing cryptographic algorithms and the proposed algorithm.

Cryptography is such a field , that every possible algorithm developed in this field provided a specific level of security. Though every successive algorithm developed , boasted about the enhanced level of security it provided, it was always at the price of certain critical factor , be it time complexity or space complexity.

Key-Enabled S-Box (KES) algorithm , is a major modification over the existing Advanced Encryption Standard(AES) algorithm which takes the input data and passes it through 4 operations namely , SubByte operation , ShiftRow operation, MixColumn operation and the AddRoundKey operation. The 2 modifications that is done over the current AES algorithm are as follows :

1. The 1st modification involves providing security to the S- Box that is used in the SubByte operation.
2. The 2nd modification deals with eliminating the MixColumn operation since the the time complexity of the AES algorithm is more due to this operation.

The reason behind the above modifications lye in the fact that if a hacker manages to decode the S- Box , it makes the hackers work really simple thus compromising the security of the system. The proposed system provides security to the S-Box by encrypting it with a round key , which doubles the work of the hacker since he has to first decode the S-Box and then decode the rest of the algorithm .

The project to be produced will be focusing on enhancing the security provided by the existing algorithm so it play a pivot role in applications such as electronic voting system , etc.

* 1. **MOTIVATION**

The AES(Advanced Encryption Standard) encryption algorithm which is quite a reliable algorithm for security purposes even till today , there are certain areas where the AES algorithm falls behind. For small files AES algorithm is pretty fast in encypting information, but while encrypting big files, it ends up taking a lot of time which is not feasible for modern day applications . Another drawback in the AES algorithm is that once the S-Box is decoded , it is easy to extract the encrypted information. Thus the KES algorithm deals with these drawbacks by eliminating the MixColumn operation in the AES algorithm and encrypting the S-Box too. This not only ensures double the security but also decreases the processing time to the encryption and decryption process.

**1.2 PROBLEM STATEMENT**

The iterative period of AES S-Box has short-period phenomenon, and all the periods are less than 88. In depth analysis of AES S-Box algebraic expression indicated that the algebraic expression of AES S-Box is very simple and only 9 terms are involved. An improved S-Box, and the algebraic expression of the S-Box involving 255 terms is proposed below, thus increasing the complexity of the S-Box. Fixed S-Box allows attackers to study S-Box and find weak points while by using key-dependent S-Box approach, it makes it harder for attacker to do any offline analysis of an attack of one particular set of S- boxes. The cipher structure resembles the original AES, only the S-Box is made key-dependent without changing the value.

* 1. **SCOPE OF THE PROJECT**

In today’s world , most of the communication is done using electronic media. Data security plays a vital role in such communication. Hence there is a need to protect data from malicious attacks. This can be achieved by Cryptography. The notable cryptographic algorithms used in the past such as DES & AES had certain loopholes .

The DES algorithm was susceptible due to small key size and its vulnerability to the Brute force attack. The AES algorithm has slower processing time and a vulnerable S-Box. These loopholes have been dealt with in the proposed algorithm.

**CHAPTER NO. II : LITERATURE SURVEY**

There are many algorithms that work on symmetric key cryptography that were used for encryption and decryption which are as follows:

The standard algorithms devised yet are as follows:

1. **Data Encryption Standard (DES) Algorithm**

Up until recently, the main standard for encrypting data was a symmetric algorithm

known as the Data Encryption Standard (DES). However, this has now been replaced

by a new standard known as the Advanced Encryption Standard (AES) which we

will look at later. DES is a 64 bit block cipher which means that it encrypts data 64

bits at a time. This is contrasted to a stream cipher in which only one bit at a time (or

sometimes small groups of bits such as a byte) is encrypted.

DES was the result of a research project set up by International Business Machines

(IBM) corporation in the late 1960’s which resulted in a cipher known as LUCIFER. In the early 1970’s it was decided to commercialise LUCIFER and a number of significant changes were introduced. IBM was not the only one involved in these changes as they sought technical advice from the National Security Agency (NSA) (other outside consultants were involved but it is likely that the NSA were the major contributors from a technical point of view). The altered version of LUCIFER was put forward as a proposal for the new national encryption standard requested by the National Bureau of Standards (NBS)3. It was finally adopted in 1977 as the Data Encryption Standard - DES (FIPS PUB 46).

Some of the changes made to LUCIFER have been the subject of much controversy

even to the present day. The most notable of these was the key size. LUCIFER used

a key size of 128 bits however this was reduced to 56 bits for DES. Even though DES

actually accepts a 64 bit key as input, the remaining eight bits are used for parity

checking and have no effect on DES’s security. Outsiders were convinced that the 56

bit key was an easy target for a brute force attack4 due to its extremely small size. The

need for the parity checking scheme was also questioned without satisfying answers.

Another controversial issue was that the S-Boxes used were designed under classified

conditions and no reasons for their particular design were ever given. This led people

to assume that the NSA had introduced a “trapdoor” through which they could decrypt any data encrypted by DES even without knowledge of the key. One startling discovery was that the S-Boxes appeared to be secure against an attack known as Differential Cryptanalysis which was only publicly discovered by Biham and Shamir in 1990.

This suggests that the NSA were aware of this attack in 1977; 13 years earlier, In fact

the DES designers claimed that the reason they never made the design specifications for the S-Boxes available was that they knew about a number of attacks that weren’t public knowledge at the time and they didn’t want them leaking - this is quite a plausible claim as differential cryptanalysis has shown. However, despite all this controversy, in 1994 NIST reaffirmed DES for government use for a further five years for use in areas other than “classified”.

DES of course isn’t the only symmetric cipher. There are many others, each with varying levels of complexity. Such ciphers include: IDEA, RC4, RC5, RC6 and the new Advanced Encryption Standard (AES). AES is an important algorithm and was originally meant to replace DES (and its more secure variant triple DES) as the standard algorithm for *non-classified* material. However as of 2003, AES with key sizes of 192 and 256 bits has been found to be secure enough to protect information up to *top secret*.

Since its creation, AES had underdone intense scrutiny as one would expect for

an algorithm that is to be used as the standard. To date it has withstood all attacks but

the search is still on and it remains to be seen whether or not this will last. We will

look at AES later in the course.

**Inner workings of DES**

DES (and most of the other major symmetric ciphers) is based on a cipher known as

the Feistel block cipher. This was a block cipher developed by the IBM cryptography

researcher Horst Feistel in the early 70’s. It consists of a number of rounds where

each round contains bit-shuffling, non-linear substitutions (S-Boxes) and exclusive OR operations. Most symmetric encryption schemes today are based on this structure

(known as a feistel network).

As with most encryption schemes, DES expects two inputs - the plaintext to be encrypted and the secret key. The manner in which the plaintext is accepted, and the key arrangement used for encryption and decryption, both determine the type of cipher it is. DES is therefore a symmetric, 64 bit block cipher as it uses the same key for both encryption and decryption and only operates on 64 bit blocks of data at a time5 (be they plaintext or ciphertext). The key size used is 56 bits, however a 64 bit (or eight-byte) key is actually input. The least significant bit of each byte is either used for parity (odd for DES) or set arbitrarily and does not increase the security in any way. All blocks are numbered from left to right which makes the eight bit of each byte the parity bit.

Once a plain-text message is received to be encrypted, it is arranged into 64 bit blocks

required for input. If the number of bits in the message is not evenly divisible by 64,

then the last block will be padded. Multiple permutations and substitutions are incorporated throughout in order to increase the difficulty of performing a cryptanalysis on the cipher. However, it is generally accepted that the initial and final permutations offer little or no contribution to the security of DES and in fact some software implementations omit them (although strictly speaking these are not DES as they do not adhere to the standard).

1. **Advanced Encryption Standard (AES) Algorithm –** The AES is a symmetric key encryption algorithm. It uses block size of 128, 192, or 256 bits which depend on the number of rounds. It uses the concept of ‘state’ which is made of 16 bytes .The state can be represented as a 4x4 matrix then a column transposition is applied on a block. It is efficient in terms of software and hardware utilization.
   1. **EXISTING SYSTEM**

AES is based on a design principle known as a [substitution-permutation](http://en.wikipedia.org/wiki/Substitution-permutation_network) network, combination of both substitution and permutation, and is efficient in terms of software and hardware utilization. Unlike its predecessor DES, AES does not use a [Feistel network](http://en.wikipedia.org/wiki/Feistel_network). AES is a variant of Rijndael which has a fixed [block size](http://en.wikipedia.org/wiki/Block_size_%28cryptography%29) of 128 [bits](http://en.wikipedia.org/wiki/Bit), and a [key size](http://en.wikipedia.org/wiki/Key_size) of 128, 192, or 256 bits. By contrast, the Rijndael specification as per it is specified with block and key sizes that may be any multiple of 32 bits, both with a minimum of 128 and a maximum of 256 bits.

AES operates on a 4×4 [column-major order](http://en.wikipedia.org/wiki/Column-major_order) matrix of bytes, termed the state, although some versions of Rijndael have a larger block size and have additional columns in the state. Most AES calculations are done in a special [finite field](http://en.wikipedia.org/wiki/Finite_field_arithmetic).

The key size used for an AES cipher specifies the number of repetitions of transformation rounds that convert the input, called the plaintext, into the final output, called the ciphertext. The number of cycles of repetition is as follows

* 10 cycles of repetition for 128-bit keys.
* 12 cycles of repetition for 192-bit keys.
* 14 cycles of repetition for 256-bit keys.

Each round consists of several processing steps, each containing four similar but different stages, including one that depends on the encryption key itself. A set of reverse rounds are applied to transform ciphertext back into the original plaintext using the same encryption key.

**High-level description of the algorithm**

1. KeyExpansion - Round keys are derived from the cipher key using [Rijndael's key schedule](http://en.wikipedia.org/wiki/Rijndael_key_schedule). AES requires a separate 128-bit round key block for each round plus one more.
2. InitialRound
   1. AddRoundKey - Each byte of the state is combined with a block of the round key using bitwise XOR.
3. Rounds
   1. SubBytes - A non-linear substitution step where each byte is replaced with another according to a [lookup table](http://en.wikipedia.org/wiki/Rijndael_S-box).
   2. ShiftRows – A transposition step where the last three rows of the state are shifted cyclically a certain number of steps.
   3. MixColumns - A mixing operation which operates on the columns of the state, combining the four bytes in each column.
   4. AddRoundKey
4. Final Round (no MixColumns)
   1. SubBytes
   2. ShiftRows
   3. AddRoundKey.

### The SubBytes step

### 

### *Fig.1.1: In SubBytes step, each byte in the state is replaced with its entry in a fixed 8-bit lookup table, S(bij) = S(aij)*

In the SubBytes step, each byte ai,j in the *state* matrix is replaced with a SubByte S(ai,j)using an 8-bit [substitution box](http://en.wikipedia.org/wiki/Substitution_box), the [Rijndael S-Box](http://en.wikipedia.org/wiki/Rijndael_S-box). This operation provides the non-linearity in the [cipher](http://en.wikipedia.org/wiki/Cipher). The S-Box used is derived from the [multiplicative inverse](http://en.wikipedia.org/wiki/Multiplicative_inverse) over [**GF**](http://en.wikipedia.org/wiki/Finite_field)(*28*), known to have good non-linearity properties. To avoid attacks based on simple algebraic properties, the S-Box is constructed by combining the inverse function with an invertible [affine transformation](http://en.wikipedia.org/wiki/Affine_transformation). The S-Box is also chosen to avoid any fixed points (and so is a [derangement](http://en.wikipedia.org/wiki/Derangement)), i.e., S(aij)!=ai,j ,and also any opposite fixed points, i.e., S(ai,j) ⊕ ai,j != 0xFF . While performing the decryption, Inverse SubBytes step is used, which requires first taking the affine transformation and then finding the multiplicative inverse (just reversing the steps used in SubBytes step).

### The ShiftRows step

### D:\admin2\Pictures\fig.2.jpg

### Fig.1.2: *In the ShiftRows step, bytes in each row of the state are shifted cyclically to the left. The number of places each byte is shifted differs for each row.*

The ShiftRows step operates on the rows of the state; it cyclically shifts the bytes in each row by a certain [offset](http://en.wikipedia.org/wiki/Offset_%28computer_science%29). For AES, the first row is left unchanged. Each byte of the second row is shifted one to the left. Similarly, the third and fourth rows are shifted by offsets of two and three respectively. For blocks of sizes 128 bits and 192 bits, the shifting pattern is the same. Row n is shifted left circular by n-1 bytes. In this way, each column of the output state of the ShiftRows step is composed of bytes from each column of the input state. (Rijndael variants with a larger block size have slightly different offsets). For a 256-bit block, the first row is unchanged and the shifting for the second, third and fourth row is 1 byte, 3 bytes and 4 bytes respectively—this change only applies for the Rijndael cipher when used with a 256-bit block, as AES does not use 256-bit blocks. The importance of this step is to avoid the columns being linearly independent, in which case, AES degenerates into four independent block ciphers

### The MixColumns step

### D:\admin2\Pictures\fig3.jpg

### Fig.1.3 : *In the MixColumns step, each column of the state is multiplied with a fixed polynomial c(x).*

In the MixColumns step, the four bytes of each column of the state are combined using an invertible [linear transformation](http://en.wikipedia.org/wiki/Linear_transformation). The MixColumns function takes four bytes as input and outputs four bytes, where each input byte affects all four output bytes. Together with ShiftRows, MixColumns provides [diffusion](http://en.wikipedia.org/wiki/Diffusion_%28cryptography%29) in the cipher.

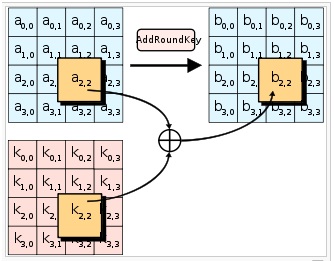
During this operation, each column is multiplied by a fixed matrix:


\begin{bmatrix}
2 & 3 & 1 & 1 \\
1 & 2 & 3 & 1 \\
1 & 1 & 2 & 3 \\
3 & 1 & 1 & 2
\end{bmatrix}


Matrix multiplication is composed of multiplication and addition of the entries, and here the multiplication operation can be defined as this: multiplication by 1 means no change, multiplication by 2 means shifting to the left, and multiplication by 3 means shifting to the left and then performing [XOR](http://en.wikipedia.org/wiki/Exclusive_or) with the initial unshifted value. After shifting, a conditional [XOR](http://en.wikipedia.org/wiki/Exclusive_or) with 0x1B should be performed if the shifted value is larger than 0xFF. (These are special cases of the usual multiplication in **GF**(*28*).) Addition is simply XOR.

In more general sense, each column is treated as a polynomial over **GF**(*28*) and is then multiplied modulo x4+1 with a fixed polynomial c(x) = 0x03 · x3 + x2 + x + 0x02. The coefficients are displayed in their [hexadecimal](http://en.wikipedia.org/wiki/Hexadecimal) equivalent of the binary representation of bit polynomials from **GF**(2)[x]. The MixColumns step can also be viewed as a multiplication by the shown particular [MDS matrix](http://en.wikipedia.org/wiki/MDS_matrix) in the [finite field](http://en.wikipedia.org/wiki/Finite_field) **GF**(*28*). This process is described further in the article [Rijndael mix columns](http://en.wikipedia.org/wiki/Rijndael_mix_columns).

### The AddRoundKey step



**Fig.1.4*:*** *In the AddRoundKey step, each byte of the state is combined with a byte of the round subkey using the* [*XOR*](http://en.wikipedia.org/wiki/Exclusive_or) *operation (⊕).*

In the AddRoundKey step, the subkey is combined with the state. For each round, a subkey is derived from the main [key](http://en.wikipedia.org/wiki/Key_%28cryptography%29) using [Rijndael's key schedule](http://en.wikipedia.org/wiki/Rijndael_key_schedule); each subkey is the same size as the state. The subkey is added by combining each byte of the state with the corresponding byte of the subkey using bitwise [XOR](http://en.wikipedia.org/wiki/Exclusive_or).

**CHAPTER NO. III : SOFTWARE REQUIREMENT SPECIFICATION**

# 1. Introduction.

## 1.1 Purpose

* 1. The purpose of the KES Encryption Algorithm is to provide the following :

1. *Authentication* – Providing ones identity before granting access.
2. *Privacy & Confidentiality* – Ensure that outsiders cannot read data intended for specific parties.
3. *Integrity* – Ensures that the message is not modified in any way before it arrives to intended recipient.
4. *Non-Repudiation* – Ensuring that the message truly originates from the sender.

## 1.2 Scope

* The scope of the project is to provide faster encryption and decryption with the help of the proposed algorithm.
* Generation of graphs for the comparative analysis of the proposed algorithm, AES algorithm and the DES algorithm based on their time complexity.

## 1.3 Definitions, Acronyms, and Abbreviations

**1.3.1 Definitions**

**ENCRYPTION:**

The process of converting plaintext into a something which is unreadable , except for the receiver is known as encryption.

**DECRYPTION:**

The process of converting cipher text(unreadable form) back to plaintext is called decryption.

**S- BOX:**

The basic component of a symmetric key algorithm which performs substitution.

**ACTOR:**

actor

A use case diagram shows the interaction between the system and entities external to the system. These external entities are referred to as actors. Actors represent roles which may include human users, external hardware or other systems.

**USE CASE:**

http://res.dotnetcoders.com/images/uml/usecase-usecase.png

Use Cases describes the behavior of the system as seen from an actor’s point of view . Use Cases are depicted with an ellipse.

|  |
| --- |
|  |

**System boundary:**

<<subsystem>>

A system boundary defines the scope of what a system will be. A system cannot have infinite functionality . A system boundary of a use case diagram defines the limits of the system. The system boundary is represented with a rectangle diagram.

**Association:**

Associations are used to link Actors with Use Cases, and indicate that an Actor participates in the Use Case in some form. Associations are depicted by a line connecting the Actor and the Use Case.

**Aggregation:**

A special form of association that are used to represent a wide range of connections among set of object.Is is denoted by a simple line with a diamond at the container end of the association.

**Generalization:**

Generalization is used when you find two or more use cases that have commonalities in behavior, structure, and purpose. When this happens, you can describe the shared parts in a new, often abstract, use case that is then specialized by child use cases.

**Include:**

*<<include>>*

It uses to include use cases that exibit additional features.

**Extend :**

*<<extend>>*

It is used for specifying exceptional behavior.

**Class :**

Name

Attribute

Operations

Each class is a collection of data and functions that manipulate the data. Each class is represented by rectangle sub-divided into 3 categories followed by name, attributes and operations.

**Compositions:**

It is a strong form of association. The lifetime of the part is dependent upon the whole.

**Object:**

Objects are entities that encapsulate state and behavior. Each object has an entity and is distinguishable from other objects.

**Lifeline:**

Lifeline is a time span of the object. They are vertical dash line that represents the existence of an object at a particular time.

**Focus of control:**

Active procedure on which object is active. It is depicted by vertical rectangle.

**Types of message:**

***Synchronous:***

The sender waits until the responder finishes.

***Asynchronous:***

The sender does not wait for anything from the responder, but it continues its own activity.

***Flat:***

The sender does not wait for anything from the responder and finishes its activity, the control is then passed to the responder.

**State:**

A simple state with no substructure.

**Initial Node:**

Start state. Indicates the starting state of the system.

**Final Node:**

Final State. Indicates the end of the activity

**Fork/Join:**

Fork represents that many activites can be parallely carried out.

**Swimlanes:**

Diagrams are partitioned according to the class who is responsible for carrying out the activities .They are represented as rectangles enclosing a group of actions.

**1.3.2 Abbreviations**

1. AES – Advanced Encryption Standard
2. DES – Data Encryption Standard
3. S-Box – Substitution Box

**1.4 References**

[1] Daemen, J., and Rijmen, V.: The block cipher Rijndael. Proceedings of the Third International Conference on smart card Reseacrh and Applications, CARDIS'98, 1820, pp. 277-284, Berlin: Springer, (2000).

[2] Stallings, W.: Cryptography and Network Security, Prentice Hall, (2010).

[3] Daemen, J. and Rijmen, V.: The First 10 Years of Advanced Encryption. In IEEE Security and Privacy, vol. 8, pp. 72-74, November (2010).

[4] Federal Information Processing Standards Publications FIPS 197, Advanced Encryption Standard (AES), 26 Nov (2001).

[5] Fahmy, A., Shaarawy, M., El-Hadad, K., Salama, G., and Hassanain, K.: A Proposal For A Key-Dependent AES. 3rd International Conference: Sciences of Electronic, Technologies of Information and Telecommunications. Tunisia: SETIT (2005).

[6] Sagheer, A. M., Al-Rawi, S. S., Dawood., A. O.: Proposing of Developed Advance encryption Standard. Developments in E-systems Engineering (DeSE), pp197-202 (2011).

[7] Zhang, R., and Zhen, L.: A Block Cipher using key-dependent S-Box and P-Box. ISIE 2008 IEEE International Symposium on Industrial Electronics, pp. 1463-1468 (2008).

[8] Krishnamurthy, G. N., and Ramasvamy, V.: Making AES Stronger:AES with Key Dependent S-Box. IJCSNS International Journal of Computer Science and Network Security, vol.8, pp. 388-398 (2008).

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

[10] Rukhin, A., et al.: A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. NIST Special Publication 800-22 (2001).

## 1.5 Overview

* This Software Requirements Specification (SRS) is the requirements work product that formally specifies Encryption and Decryption using KES Encryption Algorithm which is the modified version of the AES Algorithm.
* Requirements statements in this document are both functional and non-functional.
* These include a general description of the product, user characteristics, and general constraints of this system.
* This model demonstrates the development team's understanding of the business domain and serves to maximize the team's ability to build a system that truly does support the business.
* These requirements will allow the user to gain knowledge , as to how different encryption algorithms differ from each other in terms of providing security.

# 2. Overall Description

## 2.1 Product Perspective

* The software will run as a standalone application.
* While the software is running, the user shall be able to select which component to input values. This incorporates the selection of any plaintext producing its resultant cipher text

**2.1.1 System Interfaces**

The system is intended to interface with a single user at any given time. The user is able to interact with the system using a graphical user interface.

**2.1.2 User Interfaces**

The user interface must provide the user an understandable and effective way for entering attributes into the system. Java will be used to create the graphical user interface for the system. At the top of the GUI is a drop-down panel that will display each of the encryption algorithms that may be used (including the DES algorithm, AES algorithm & the KES Encryption algorithm). Below that is a text field that allows the user to enter the plaintext that is to be encrypted. Depending on the type of algorithm chosen, the user will also be given the option of choosing various options such as the various key or substitutions. At the bottom of the interface is a text area that displays the result of the cryptographic scheme or cipher text. Occupying the right side is a console area that displays any output that will attempt to give the user some insight into how the encryption schemes work.

## 2.2 Product Functions

* Encryption converts the input data in the garbled form so that no intruder can understand that form of data unless he/she is in possession of the secret key.
* Decryption converts the encoded message back to the plaintext.
* Generation of graphs based on the performance of the DES algorithm , AES algorithm and KES algorithm.

## 2.4 General Constraints:

* **Data**
* The maximum input block size is 128 bits.
* The key size can be either 128,192 or 256 bit only.
* **Language**:

The application and the algorithm will be designed using JAVA language.

# 3. Specific Requirements

# 3.1 External Interface Requirements

### 3.1.1 User Interfaces

* Keyboard
* Mouse

### 

**3.1.2 Encryption Algorithms**

The system must be able to allow the user to select from the following encryption schemes:

DES algorithm, AES algorithm, KES Encryption algorithm.

1. **Data Encryption Standard (DES)**

Each user will be allowed to enter in their own 64-bit key that will be used.

1. **Advanced Encryption Standard (AES)**

After the application of the key by the user , the data will go through 5 steps i.e. Byte Substitution, Row Shifting , Column Mixing & Round Key Addition for encryption . While decrypting the data , the column mixing step is eliminated in the final round.

1. **Proposed Algorithm**

In the proposed algorithm we will be eliminating the column mixing step for all the rounds and we shall be making the S-Box key dependant.

## 3.2 Functional Requirements

**3.2.1 Plaintext Input**

The Input given by the user is a plaintext message , which will be encoded by our software.

**3.2.2 Viewing Messages**

**3.2.3.1 Plaintext**

The plaintext message is viewed that the user shall input.

**3.2.3.2 Ciphertext**

The encoded ciphertext is viewed by the user in one of a variety of forms that are used in encodings today, mainly the ASCII standard, decimal, hexadecimal, and binary.

**3.2.3 View Console**

The messages will be displayed to the user which will give them an insight into the inner workings of the algorithm.

## 3.3 Non-Functional Requirements

**3.3.1 User Interface and Human Factors**

Though the user shall access this system via a standalone application, the system will still function independently. All input fields will be aligned in the center of any panel that contains the component for ease of use reasons.

**3.3.2 Deployment Platform**

**3.3.2.1 Software**

Since the system runs as a standalone application, the user must have a recent version

of Java2 installed on their system.

**3.3.2.1.1 Operating System**

Because Java is used, the operating system will be irrelevant. The system will run on any OS that supports Java2, including, but not limited to, Windows, Linux, and MacOS.

**3.3.2.1.2 Graphics**

A graphical user interface is used to enhance usability and must be able to run

on display that supports 640x480 resolution or higher

**3.3.2.2 Hardware**

The system must run on all systems that support Java2.

**3.3.3 Error Handling**

The system must check to ensure that all values in fields are valid (if an input field expects a letter of the alphabet, then great pains must be taken to ensure that the user does not input a non-alphabetic character), and that numerical fields have values that lie within the specified range.

**3.3.4 Quality Issues**

The program should not halt or crash during execution. The program must not alter the state of the user’s system after closing. As far as security issues are concerned, no files on the user’s system are read or otherwise modified.

**3.4 Software System Characteristics**

### 3.4.1 Reliability

Since the algorithm used is much faster as compared to the original AES algorithm and since there are no known security flaws yet , the reliability of this software is high.

### 3.4.2 Availability

The software is available to any party wishing to access it and will be accessible via the products home page

### 3.4.3 Security

There are no known security flaws at this time.

### 3.4.4 Maintainability

The source code should be properly documented, so that new developers will be able to understand the code as easily as possible

### 3.4.5 Portability

Since Java is used as the development language, there should be few problems porting the software to new platforms. As an added bonus, as long as the platform contains a java-capable browser, the system will run as an applet.

* 1. **Supporting Information**

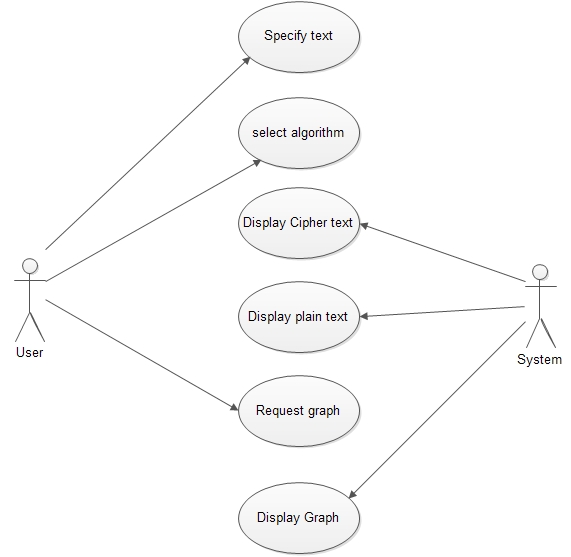
**4.1 Table of Content**

|  |  |  |
| --- | --- | --- |
| **Chapter** | **Topic** | **Page No.** |
| **1** | **Introduction** | **4** |
| 1.1 | Purpose | 4 |
| 1.2 | Scope | 4 |
| 1.3 | Definitions, Acronyms, and Abbreviations | 4 |
| 1.4 | References | 8 |
| 1.5 | Overview | 8 |
| **2** | **General Description** | 9 |
| 2.1 | Product Perspective | 9 |
| 2.2 | Product Functions | 9 |
| 2.3 | User Characteristics | 9 |
| 2.4 | General Constraints | 9 |
| **3** | **Specific Requirement** | **9** |
| **3.1** | **External Interface Requirements** | **9** |
| 3.1.1 | User Interface | 9 |
| 3.1.2 | Encryption Algorithms | 10 |
| 3.2 | **Functional Requirements** | 10 |
| 3.2.1 | Plaintext Input | 10 |
| 3.2.2 | Viewing Messages | 10 |
| 3.2.2.1 | Plaintext | 10 |
| 3.2.2.2 | Cipher text | 10 |
| 3.2.3 | View Console | 10 |
| **3.3** | **Non function Requirement** | 10 |
| 3.3.1 | * User Interface and Human Factors | 10 |
| 3.3.2 | Deployment Platform | 10 |
| 3.3.2.1 | * Software | 10 |
| 3.3.2.1.1 | Operating System | 11 |
| 3.3.2.1.2 | Graphics | 11 |
| 3.3.2.2 | Hardware | 11 |
| 3.3.3 | * Error Handling | 11 |
| 3.3.4 | Quality Issues | 11 |
| **3.4** | **Software System Characteristics** | 11 |
| 3.3.1 | Performance | 11 |
| 3.3.2 | Reliability | 11 |
| 3.3.3 | Availability | 11 |
| 3.3.4 | Security | 11 |
| 3.3.5 | Maintainability | 11 |
| 3.3.6 | Portability | 11 |

**CHAPTER NO. IV** : **REQUIREMENT GATHERING & ANALYSIS**

**4.1 Requirement Elicitation**

**Use Case Diagram and Description**

****

**Fig.4*:*** *Use Case Diagram*

|  |  |  |
| --- | --- | --- |
| **SR.NO** | **ACTORS** | **USE CASE DESCRIPTION** |
| 1 | User | * Inputs the plaintext * Selects the algorithm for encryption * Requests Graph |
| 2 | Application | * Generates Keys * Encrypts * Decrypts * Displays Encrypted Output * Generates Graphs |

**4.2 FEASIBILITY STUDY**

The very first phase in any system developing life cycle is preliminary investigation. The feasibility study is a major part of this phase. A measure of how beneficial or practical the development of any information system would be to the organization is the feasibility study.

* + 1. **TECHNICAL FEASIBILITY**
    - At least 1.2 GHz Pentium Processor or Intel compatible processor.
    - At least 256 MB RAM.
    - A mouse or other pointing device.
    - At least 10GB free hard disk space.
    - At least JAVA Software-1.7
    - NetBeans IDE 7.4

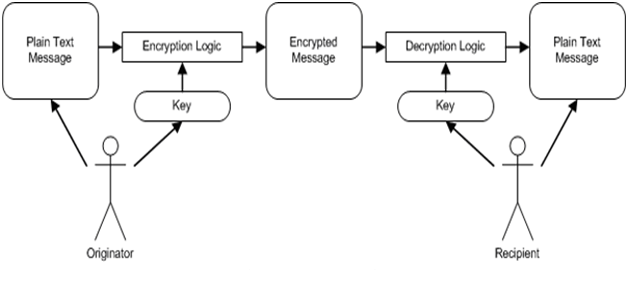
* + 1. **ECONOMICAL FEASIBILITY**

For the user, application will be economically feasible in the following aspects:

* Our application will reduce the time that is wasted in the computational operation standard algorithm which will reduce the overall cost.
* No additional hardware is required in the proposed system (i.e., it can work on any system which satisfies the minimum requirements of the proposed system).

* 1. **REQUIREMENT ANALYSIS**

**4.3.1 BLOCK DIAGRAM**

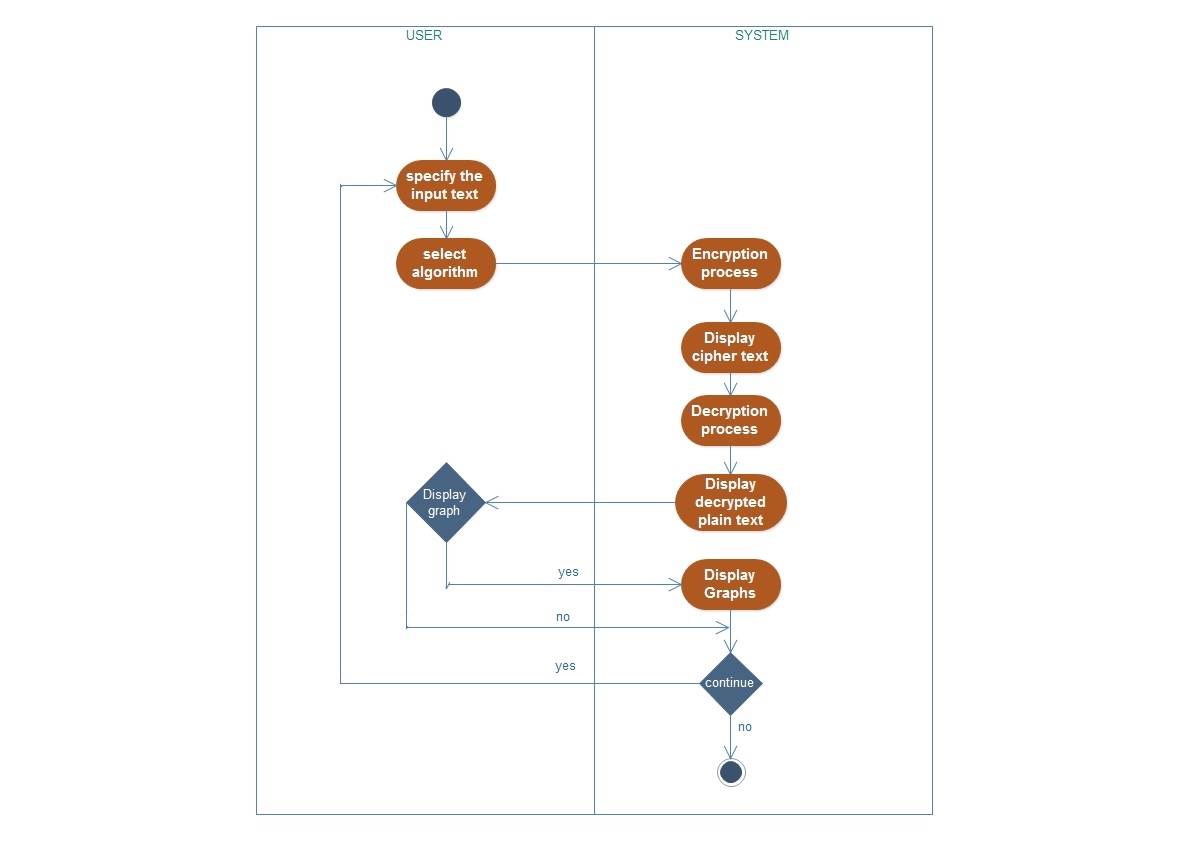
****

**Fig.5:** *Block Diagram*

**CHAPTER NO. V** : **ANALYSIS**

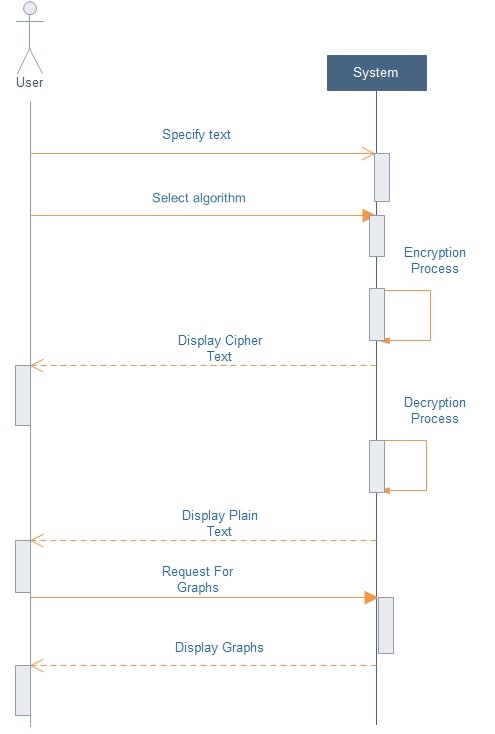
**5.1 UML Diagrams**

**Activity Diagram**

****

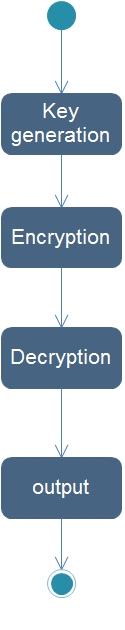
***Fig.6:*** *Activity Diagram*

**Sequence Diagram**

****

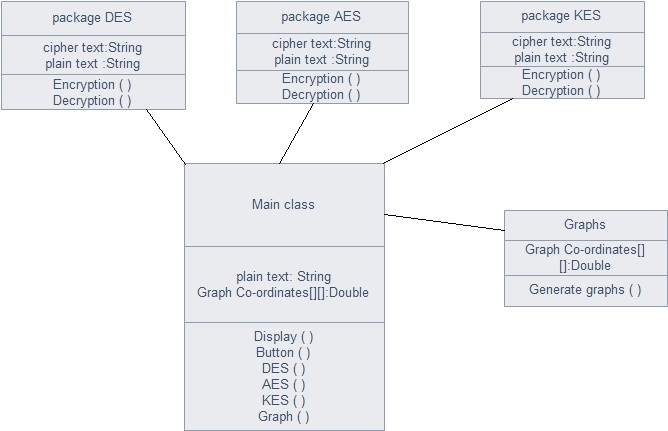
***Fig.7*** *– Sequence Diagram*

**State Diagram**

****

***Fig.8*** *– State Diagram*

**Class Diagram**

****

***Fig.9*** *– Class Diagram*

**CHAPTER NO. VI : PROPOSED SYSTEM**

The standard AES algorithm uses a predefined S-Box for performing byte substitution. The proposed algorithm improves the standard S-Box of the algorithm by making it key dependent which results in a Dynamic and secure S-Box.

The computation time of an algorithm also plays an important role in efficiency calculations. To improve the time complexity of the algorithm, the MixColumn operation has been eliminated. This makes the proposed algorithm faster than the Standard AES algorithm. At the same time the complexity of the algorithm is also intact. For any algorithm to be optimal the major requirements is to deliver the best possible security (complexity) in the least possible time. Keeping this in mind the proposed algorithm provides decent security in comparatively less time.

**High-level description of the algorithm**

1. **KeyExpansion -** Round keys are derived from the cipher key using [Rijndael's key schedule](http://en.wikipedia.org/wiki/Rijndael_key_schedule). AES requires a separate 128-bit round key block for each round plus one more.
2. **S-Box Encryption** –In this step the each value of the S-Box is XORed with the value obtained after all the 16 round key values go through the XOR operation.
3. **Rounds** 
   * 1. **SubBytes** - This stage (known as SubBytes) is simply a table lookup using a 16×16 matrix of byte values called an S-Box. This matrix consists of all the possible combinations of an 8 bit sequence However, the S-Box is not just a random permutation of these values and there is a well defined method for creating the S-Box tables. Again the matrix that gets operated upon throughout the encryption is known as state. For this particular round each byte is mapped into a new byte in the following way: the leftmost nibble of the byte is used to specify a particular row of the S-Box and the rightmost nibble specifies a column. The Inverse substitute byte transformation (known as InvSubBytes) makes use of an inverse S-Box.
     2. **ShiftRows –** This stage (known as ShiftRows). This is a simple permutation . It works as follow:

* The first row of state is not altered.
* The second row is shifted 1 bytes to the left in a circular manner.
* The third row is shifted 2 bytes to the left in a circular manner.
* The fourth row is shifted 3 bytes to the left in a circular manner.

The Inverse Shift Rows transformation (known as InvShiftRows) performs these circular

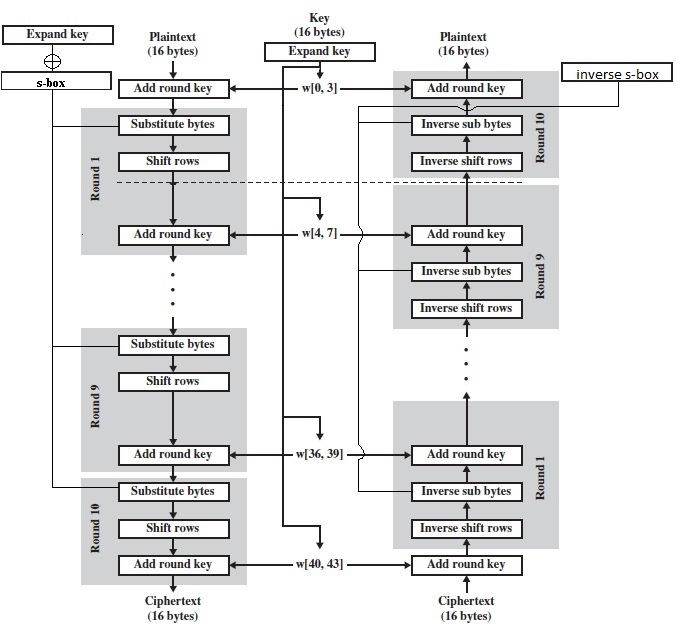
shifts in the opposite direction for each of the last three rows (the first row was

unaltered to begin with).

This operation may not appear to do much but if you think about how the bytes are

ordered within state then it can be seen to have far more of an impact. A one byte shift is therefore a linear distance of four bytes. The transformation also ensures that the four bytes of one column are spread out to four different columns.

* + 1. **AddRoundKey-**In this stage (known as AddRoundKey) the 128 bits of state are bitwise XORed with the 128 bits of the round key. The operation is viewed as a columnwise operation between the 4 bytes of a state column and one word of the round key. This transformation is as simple as possible which helps in efficiency but it also effects every bit of state.



**Fig.2*:*** *KES Architecture Diagram*

**CHAPTER NO. VI : IMPLEMENTATION**

**7.1 ALGORITHM OR METHODOLOGY**

**7.2 TECHNOLOGY USED**

1. **1. JAVA**



Java is a computer programming language. It enables programmers to write computer instructions using English based commands, instead of having to write in numeric codes. It’s known as a “high-level” language because it can be read and written easily by humans. Like English, Java has a set of rules that determine how the instructions are written. These rules are known as its “syntax”. Once a program has been written, the high-level instructions are translated into numeric codes that computers can understand and execute.

Java was designed with a few key principles in mind:

* Easy to Use: The fundamentals of Java came from a [programming language called c++.](http://cplus.about.com/od/introductiontoprogramming/p/profileofcpp.htm) Although c++ is a powerful language, it was felt to be too complex in its syntax, and inadequate for all of Java's requirements. Java built on, and improved the ideas of c++, to provide a programming language that was powerful and simple to use.
* Reliability: Java needed to reduce the likelihood of fatal errors from programmer mistakes. With this in mind, [object-oriented programming](http://java.about.com/od/objectorientedprogramming/a/introobjects.htm) was introduced. Once data and its manipulation were packaged together in one place, it increased Java’s robustness.
* Secure: As Java was originally targeting mobile devices that would be exchanging data over networks, it was built to include a high level of security. Java is probably the most secure programming language to date.
* Platform Independent: Programs needed to work regardless of the machine they were being executed on. Java was written to be a portable language that doesn't care about the [operating system](http://cplus.about.com/od/introductiontoprogramming/a/opersys.htm) or the hardware of the computer

**7.3 PROJECT TIME LINE CHART**

