*INTRODUCTION*

**1. INTRODUCTION**

* 1. **Background of the Project**

In the era of digital transaction, in order to transact the data in a more secured manner the need for a cryptographic algorithm is inevitable. There are numerous numbers of cryptographic algorithms which makes the system invulnerable from the attacks of intruders and eavesdroppers.

Cryptography is the process of writing a secret code that is only known to the targetted users leaving the third parties with no clue what the code is about. The need for cryptographic algorithm is to protect the sensitive data from unauthorised usage. Considering communication between applications, there are some specific security speculations such as authentication, privacy/confidentiality, integrity, non-repudiation. In order to meet the above speculations, various cryptographic algorithms are being used. Thus security is achieved using strong cryptographic algorithms.

In cryptography, a **key** is a piece of information (a parameter) that determines the functional output of a cryptographic algorithm or cipher. Without a key, the algorithm would produce no useful result. In encryption, a key specifies the particular transformation of plaintext into cipher text, or vice versa during decryption. A key is used to encrypt and decrypt whatever data is being encrypted / decrypted. Keys are also used in other cryptographic algorithms, such as digital signature schemes and message authentication codes. Key generation is the process of generating keys for cryptography. To prevent a key from being guessed, keys need to be generated truly randomly and contain sufficient entropy.

In this project, we propose a software toolkit for increasing the key strength so that it will be very hard for the intruder to break the key and this technique will act as black box so that intruder will have no idea about the key formation. So it will lead to increased key complexity which obviously results the intruder nothing else than confusion and frustration.

*SOFTWARE PROJECT PLAN*

**2. SOFTWARE PROJECT PLAN**

**2.1 Existing System**

The existing system comprises of a black box technique for generating a new complex key from the key provided initially by the user. The new complex key thus obtained is hard to crack because of the complex black box design. In general black box may be of anything which has a set of confused and diffused mathematical formulas.

**2.2 Proposed System**

In this project we introduce a new black box technique for generating a complex key from the user given key. The new key is very hard to crack by the intruders because of the complexity in our proposed black box design. Here we use the Magic Square concept for designing the black box. The objective of our project is to improve the effectiveness of any cryptographic algorithm by generating a strong key. Since the new key is generated dynamically and randomly after the input of the user’s key, it is difficult for the intruders and the eve-droppers to crack the system.

**2.3 Modules**

1. Magic Square Construction

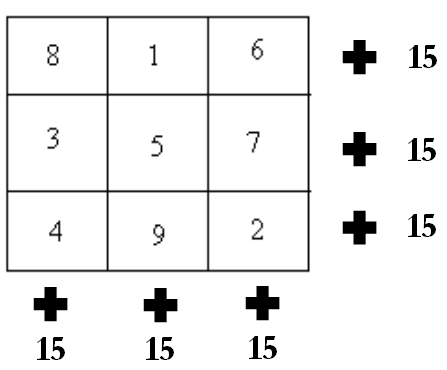
2. Strengthened Key Extraction

3. Encryption

4. Variations in Magic Square Construction

***2.3.1 Magic Square Construction***

Magic Square is a matrix of size n where the numbers from 1 to in the matrix in a manner such a way that the sum when done in the row-side or column-side or diagonal-side gives a sum of. The initial magic square is constructed using the user given key. The empty cell of magic square are filled with characters in a specific order.

******

***2.3.2 Strengthened Key Extraction***

The strengthened key is extracted from the constructed magic square by traversing through the matrix in different orders such as row-wise or column-wise. The variations include traversals such as depth-first, breadth-first, zig-zag, maze etc.

***2.3.3 Encryption***

Once the strengthened key is extracted, it is fed as an input to any cryptographic algorithm which is to be enhanced in terms of security. The cipher text obtained from the encryption is highly secured in a way such that it makes the intruders almost impossible to crack the data transmitted.

***2.3.4 Variations in Magic Square Construction***

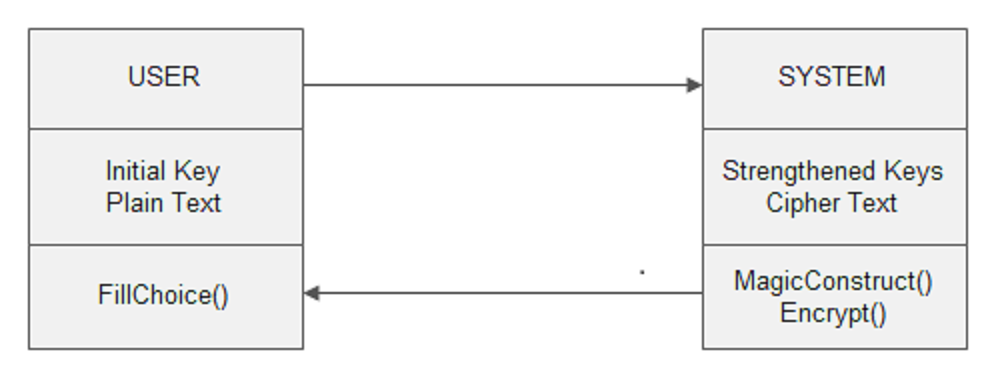
The basic method of constructing magic square includes filling the empty cells using alphabets from A to Z or in reverse order. The other variations include filling the empty cells with numeric characters or complex manipulations such as XOR of two magic squares.

*SOFTWARE REQUIREMENT SPECIFICATION*

**3. SOFTWARE REQUIREMENT SPECIFICATION**

**3.1 Functional Requirements**

***3.1.1 Class Diagram***

******

*Fig. no: 3.1.2 Class Diagram for user and system.*

***3.1.2 Front End Application***

In this project, Command Prompt is used to get the initial key from the user, choice of filling empty cells in magic square and the input file to be encrypted.

**3.2 NON FUNCTIONAL REQUIREMENTS**

***3.2.1 Performance Requirements***

The objective is to provide an efficient way of encryption to improve security in digital transmission of data. The user given key is strengthened and used as input for any cryptographic algorithm. The output of encryption, which is the cipher text, is stored in form of files. This file is again used for decryption of the cipher text to obtain the original message or the plain text.

***3.2.2 Interface Requirements***

The command prompt is used as an interface between the user and system where the user enters the initial key and the input file or text to be encrypted is given.

***3.2.3 Resource Requirements***

* *Software Requirements:*

Operating System - Windows

Front End - Java Applet

Language - Java

* *Hardware Requirements:*

Processor - Pentium IV or more

RAM - 512 Mb or more

Hard Disk - 50GB or Higher

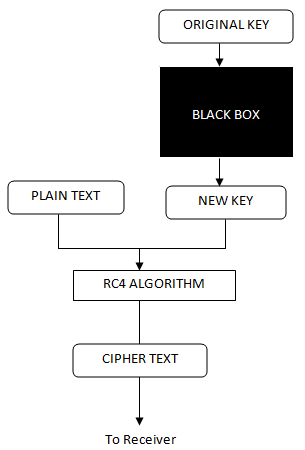
Monitor - Display Panel(640 x 480)

*SYSTEM ANALYSIS*

**4. SYSTEM ANALYSIS**

**4.1 Dataflow Diagram**

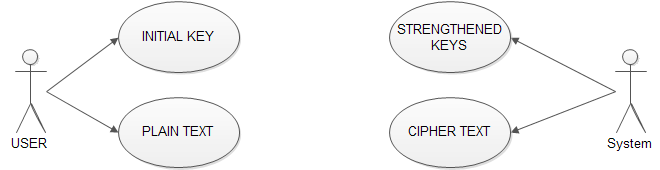
A data-flow diagram (DFD) is a graphical representation of the “flow” of data through an information system. DFDs can also be used for the visualization of data processing.



*Fig. no: 4.1.1 Dataflow diagram.*

**4.2 Use Case Diagram**

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals, and any dependencies between those use cases.



*Fig. no: 4.2.1 Use Case diagram*

*SYSTEM DESIGN*

**5. SYSTEM DESIGN**

**5.1 Front End Design**

***5.1.1. Applet***

A Javaapplet is an applet delivered to users in the form of Java byte code. Java applets can be part of a web page and executed by the Java Virtual Machine (JVM) in a process separate from the web browser, or run in Sun's Applet Viewer, a stand-alone tool for testing applets. Java applets were introduced in the first version of the Java language in 1995, and are written in programming languages that compile to Java byte code, usually in Java, but also in other languages such as Python, Ruby, or Eiffel

Java applets run at very fast speeds comparable to, but generally slower than, other compiled languages such as C++, but until approximately 2011 many times faster than JavaScript. In addition they can use 3D hardware acceleration that is available from Java. This makes applets well suited for non-trivial, computation intensive visualizations. As browsers have gained support for hardware accelerated graphics thanks to the canvas technology (or specifically WebGL in the case of 3D graphics), as well as just in time compiled JavaScript, the speed difference has become less noticeable.

Since Java's byte code is cross-platform or platform independent, Java applets can be executed by browsers for many platforms, including Microsoft Windows, UNIX, OS X and Linux. It is also trivial to run a Java applet as application software with very little extra code so that it can be run directly from the integrated development environment (IDE).

Applets are used to provide interactive features to web applications that cannot be provided by HTML alone. They can capture mouse input and also have controls like buttons or check boxes. In response to the user action an applet can change the provided graphic content. This makes applets well suitable for demonstration, visualization and teaching. There are online applet collections for studying various subjects, from physics to heart physiology.Applets are also used to create online game collections that allow players to compete against live opponents in real-time.

An applet can also be a text area only, providing, for instance, a cross platform command-line interface to some remote system. If needed, an applet can leave the dedicated area and run as a separate window. However, applets have very little control over web page content outside the applet dedicated area, so they are less useful for improving the site appearance in general (while applets like news tickersor WYSIWYG editorsare also known). Applets can also play media in formats that are not natively supported by the browser.

HTML pages may embed parameters that are passed to the applet. Hence the same applet may appear differently depending on the parameters that were passed. As applets have been available before CSS, they were also widely used for trivial effects like navigation buttons. This use is criticized and declining

***5.1.2. Java (Programming Language)***

Java is a general-purpose, concurrent, class-based, object-oriented computer programming language that is specifically designed to have as few implementation dependencies as possible. It is intended to let application developers "write once, run anywhere" (WORA), meaning that code that runs on one platform does not need to be recompiled to run on another. Java applications are typically compiled to byte code (class file) that can run on any Java virtual machine (JVM) regardless of computer architecture. Java is, as of 2012, one of the most popular programming languages in use, particularly for client-server web applications, with a reported 10 million users.Java was originally developed by James Gosling at Sun Microsystems (which has since merged into Oracle Corporation) and released in 1995 as a core component of Sun Microsystems' Java platform. The language derives much of its syntax from C and C++, but it has fewer low-level facilities than either of them.

The original and reference implementation Java compilers, virtual machines, and class libraries were developed by Sun from 1991 and first released in 1995. As of May 2007, in compliance with the specifications of the Java Community Process, Sun relicensed most of its Java technologies under the GNU General Public License. Others have also developed alternative implementations of these Sun technologies, such as the GNU Compiler for Java and GNU Class path.

***5.1.3. Versions***

* JDK 1.0 (January 21, 1996)
* JDK 1.1 (February 19, 1997)
* J2SE 1.2 (December 8, 1998)
* J2SE 1.3 (May 8, 2000)
* J2SE 1.4 (February 6, 2002)
* J2SE 5.0 (September 30, 2004)
* Java SE 6 (December 11, 2006)
* Java SE 7 (July 28, 2011)

***5.1.4. Java platform***

One characteristic of Java is portability, which means that computer programs written in the Java language must run similarly on any hardware/operating-system platform. This is achieved by compiling the Java language code to an intermediate representation called Java byte code, instead of directly to platform-specific machine code. Java byte code instructions are analogous to machine code, but they are intended to be interpreted by a virtual machine (VM) written specifically for the host hardware. End-users commonly use a Java Runtime Environment (JRE) installed on their own machine for standalone Java applications, or in a Web browser for Java applets.

Standardized libraries provide a generic way to access host-specific features such as graphics, threading, and networking.

A major benefit of using byte code is porting. However, the overhead of interpretation means that interpreted programs almost always run more slowly than programs compiled to native executables would. Just-in-Time (JIT) compilers were introduced from an early stage that compiles byte codes to machine code during runtime.

***5.1.5. Implementations***

Oracle Corporation is the current owner of the official implementation of the Java SE platform, following their acquisition of Sun Microsystems on January 27, 2010. This implementation is based on the original implementation of Java by Sun. The Oracle implementation is available for Mac OS X, Windows and Solaris. Because Java lacks any formal standardization recognized by Ecma International, ISO/IEC, ANSI, or other third-party standards organization, the Oracle implementation is the de facto standard.

The Oracle implementation is packaged into two different distributions: The Java Runtime Environment (JRE) which contains the parts of the Java SE platform required to run Java programs and is intended for end-users, and the Java Development Kit (JDK), which is intended for software developers and includes development tools such as the Java compiler, Javadoc, Jar, and a debugger.

OpenJDK is another notable Java SE implementation that is licensed under the GPL. The implementation started when Sun began releasing the Java source code under the GPL. As of Java SE 7, OpenJDK is the official Java reference implementation.

The goal of Java is to make all implementations of Java compatible. Historically, Sun's trademark license for usage of the Java brand insists that all implementations be "compatible". This resulted in a legal dispute with Microsoft after Sun claimed that the Microsoft implementation did not support RMI or JNI and had added platform-specific features of their own. Sun sued in 1997 and in 2001 won a settlement of US$20 million, as well as a court order enforcing the terms of the license from Sun. As a result, Microsoft no longer ships Windows with Java. Platform-independent Java is essential to Java EE, and an even more rigorous validation is required to certify an implementation. This environment enables portable server-side applications.

***5.1.6. Performance***

Programs written in Java have a reputation for being slower and requiring more memory than those written in C++. However, Java programs' execution speed improved significantly with the introduction of Just-in-time compilation in 1997/1998 for Java 1.1,the addition of language features supporting better code analysis (such as inner classes, the StringBuffer class, optional assertions, etc.), and optimizations in the Java virtual machine itself, such as HotSpot becoming the default for Sun's JVM in 2000. As of December 2012, microbenchmarks show Java 7 is approximately 44% slower than C++.Some platforms offer direct hardware support for Java; there are microcontrollers that can run Java in hardware instead of a software Java virtual machine, and ARM based processors can have hardware support for executing Java byte code through their Jazelle option.

***5.1.7. Automatic memory management***

Java uses an automatic garbage collector to manage memory in the object lifecycle. The programmer determines when objects are created, and the Java runtime is responsible for recovering the memory once objects are no longer in use. Once no references to an object remain, the unreachable memory becomes eligible to be freed automatically by the garbage collector. Something similar to a memory leak may still occur if a programmer's code holds a reference to an object that is no longer needed, typically when objects that are no longer needed are stored in containers that are still in use. If methods for a nonexistent object are called, a "null pointer exception" is thrown.

One of the ideas behind Java's automatic memory management model is that programmers can be spared the burden of having to perform manual memory management. In some languages, memory for the creation of objects is implicitly allocated on the stack, or explicitly allocated and deallocated from the heap. In the latter case the responsibility of managing memory resides with the programmer. If the program does not deallocate an object, a memory leak occurs. If the program attempts to access or deallocate memory that has already been deallocated, the result is undefined and difficult to predict, and the program is likely to become unstable and/or crash. This can be partially remedied by the use of smart pointers, but these add overhead and complexity. Note that garbage collection does not prevent "logical" memory leaks, i.e. those where the memory is still referenced but never used.

Garbage collection may happen at any time. Ideally, it will occur when a program is idle. It is guaranteed to be triggered if there is insufficient free memory on the heap to allocate a new object; this can cause a program to stall momentarily. Explicit memory management is not possible in Java.

Java does not support C/C++ style pointer arithmetic, where object addresses and unsigned integers (usually long integers) can be used interchangeably. This allows the garbage collector to relocate referenced objects and ensures type safety and security.

As in C++ and some other object-oriented languages, variables of Java's primitive data types are not objects. Values of primitive types are either stored directly in fields (for objects) or on the stack (for methods) rather than on the heap, as commonly true for objects (but see Escape analysis). This was a conscious decision by Java's designers for performance reasons. Because of this, Java was not considered to be a pure object-oriented programming language. However, as of Java 5.0, autoboxing enables programmers to proceed as if primitive types were instances of their wrapper class.

Java contains multiple types of garbage collectors. By default, HotSpot uses the Concurrent Mark Sweep collector, also known as the CMS Garbage Collector. However, there are also several other garbage collectors that can be used to manage the Heap. For 90% of applications in Java, the CMS Garbage Collector is good enough.

***5.1.8. JAVA (Software Platform)***

An edition of the Java platform is the name for a bundle of related programs from Sun that allow for developing and running programs written in the Java programming language. The platform is not specific to any one processor or operating system, but rather an execution engine (called a virtual machine) and a compiler with a set of libraries that are implemented for various hardware and operating systems so that Java programs can run identically on all of them.

* Java Card: A technology that allows small Java-based applications (applets) to be run securely on smart cards and similar small-memory devices.
* Java ME (Micro Edition): Specifies several different sets of libraries (known as profiles) for devices with limited storage, display, and power capacities. Often used to develop applications for mobile devices, PDAs, TV set-top boxes, and printers.
* Java SE (Standard Edition): For general-purpose use on desktop PCs, servers and similar devices.
* Java EE (Enterprise Edition): Java SE plus various APIs useful for multi-tier client–server enterprise applications.

The Java platform consists of several programs, each of which provides a portion of its overall capabilities. For example, the Java compiler, which converts Java source code into Java byte code (an intermediate language for the JVM), is provided as part of the Java Development Kit (JDK). The Java Runtime Environment (JRE), complementing the JVM with a just-in-time (JIT) compiler, converts intermediate byte code into native machine code on the fly. An extensive set of libraries are also part of the Java platform.

The essential components in the platform are the Java language compiler, the libraries, and the runtime environment in which Java intermediate byte code "executes" according to the rules laid out in the virtual machine specification.

***5.1.9. Java Virtual Machine***

The heart of the Java platform is the concept of a "virtual machine" that executes Java byte code programs. This byte code is the same no matter what hardware or operating system the program is running under. There is a JIT (Just in Time) compiler within the Java VirtualMachine, or JVM. The JIT compiler translates the Java byte code into native processor instructions at run-time and caches the native code in memory during execution.

The use of byte code as an intermediate language permits Java programs to run on any platform that has a virtual machine available. The use of a JIT compiler means that Java applications, after a short delay during loading and once they have "warmed up" by being all or mostly JIT-compiled, tend to run about as fast as native programs.Since JRE version 1.2, Sun's JVM implementation has included a just-in-time compiler instead of an interpreter.

Although Java programs are cross-platform or platform independent, the code of the Java Virtual Machines (JVM) that executes these programs is not. Every supported operating platform has its own JVM.

***5.1.10. Class libraries***

In most modern operating systems (OSs), a large body of reusable code is provided to simplify the programmer's job. This code is typically provided as a set of dynamically loadable libraries that applications can call at runtime. Because the Java platform is not dependent on any specific operating system, applications cannot rely on any of the pre-existing OS libraries. Instead, the Java platform provides a comprehensive set of its own standard class libraries containing much of the same reusable functions commonly found in modern operating systems. Most of the system library is also written in Java. For instance, Swing library paints the user interface and handles the events itself, eliminating many subtle differences between how different platforms handle even similar components.

The Java class libraries serve three purposes within the Java platform. First, like other standard code libraries, the Java libraries provide the programmer a well-known set of functions to perform common tasks, such as maintaining lists of items or performing complex string parsing. Second, the class libraries provide an abstract interface to tasks that would normally depend heavily on the hardware and operating system. Tasks such as network access and file access are often heavily intertwined with the distinctive implementations of each platform. The java.net and java.io libraries implement an abstraction layer in native OS code, and then provide a standard interface for the Java applications to perform those tasks. Finally, when some underlying platform does not support all of the features a Java application expects, the class libraries work to gracefully handle the absent components, either by emulation to substitute, or at least by providing a consistent way to check for the presence of a specific feature.

*CODING*

**6. CODING**

**6.1 Sample Coding**

***6.1.1 Magic Square Construction:***

BufferedReader br=new BufferedReader(newInputStreamReader(System.in));

**//Initially assigning dimension=3**

int n=3;

**//Getting the key from the User**

System.out.println("\t\tDATA ENCRYPTION");

System.out.println("Enter the key:");

String str=br.readLine();

char key[]=str.toCharArray();

**//To find the dimension of the magic square to be used based in the key length**

while(str.length()>n\*n)

{ n=n+2;

}

char box[][]=new char[n][n]; **//Initialising the magic square**

char c=65; **//ASCII value for "A"**

**//Making all elements zero**

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

box[i][j]=0;

}

}

**//Generating the key square**

int row=0;

int col=n/2;

for (int p = 0; p < n\*n; p++)

{

if(p<str.length())

{

box[row][col]=key[p];

}

else

{

box[row][col]=c;

c++;

}

row--;

col++;

if(row==-1)

row=n-1;

if(col==n)

col=0;

if(row==n-1 && col==0)

{

row=1;

col=n-1;

}

if((box[row][col]!=0)) {

row+=2;

col--;

if(row==n) row=0;

if(col==-1) col=0;

}

}

***6.1.2. Strengthened Key Extraction: (Horizontal & Vertical Keys)***

**//Generating the keys**

System.out.println("\n\nMagic Box");

String horizontal\_key=""; **//Initialising the String for storing the horizontally parsed array**

String vertical\_key=""; **//Initialising the String for storing the vertically parsed array**

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++)

{

horizontal\_key=horizontal\_key+box[i][j];

vertical\_key=vertical\_key+box[j][i];

System.out.print( box[i][j]+" ");

}

System.out.println("");

}

System.out.println("\n\nThe Newly horizontal generated key is: "+horizontal\_key);

System.out.println("\nThe Newly vertical generated key is: "+vertical\_key);

***6.1.3. Depth-First & Breadth-First Keys:***

int i=0,j=0,cnt=0;

String depth\_key="";

while(j<n){

if(i==0){

while(cnt<n){

depth\_key=depth\_key+result[i][j];

i++;

cnt++;

}

i--;

}

else{

while(cnt<n){

depth\_key=depth\_key+result[i][j];

i--;

cnt++;

}

i++;

}

j++;

cnt=0;

}

i=0; j=0;cnt=0;

String breadth\_key="";

while(i<n){

if(j==0){

while(cnt<n){

breadth\_key=breadth\_key+result[i][j];

j++;

cnt++;

}

j--;

}

else{

while(cnt<n){

breadth\_key=breadth\_key+result[i][j];

j--;

cnt++;

}

j++;

}

i++;

cnt=0;

}

***6.1.4. Zig-Zag & Maze Keys***:

**//Genertaing the zigzag key**

int i,j,sum=0,cnt=0;

String zigzag\_key="";

while(sum<n){

i=sum;

j=cnt;

while(j<=sum){

zigzag\_key=zigzag\_key+result[i][j];

i--;j++;

}

sum++;

}

sum=sum-1;

cnt++;

while(sum>=cnt){

i=sum;j=cnt;

while(j<=sum){

zigzag\_key=zigzag\_key+result[i][j];

i--;j++;

}

cnt++;

}

//**Generating the maze key**

int loop\_min=0,loop\_max=n,out\_count=0,in\_count=0;

String maze\_key="";

while(out\_count<n/2){

i=loop\_min;

j=loop\_min;

while(in\_count<loop\_max){

maze\_key=maze\_key+result[i][j];

i++;in\_count++;

}

i--;j++;

in\_count=loop\_min;

while(in\_count<loop\_max-1){

maze\_key=maze\_key+result[i][j];

j++;in\_count++;

}

j--;i--;

in\_count=loop\_min;

while(in\_count<loop\_max-1){

maze\_key=maze\_key+result[i][j];

i--;in\_count++;

}

i++;j--;

in\_count=loop\_min;

while(in\_count<loop\_max-2){

maze\_key=maze\_key+result[i][j];

j--;

in\_count++;

}

loop\_min=loop\_min+1;

loop\_max=loop\_max-1;

out\_count++;

in\_count=loop\_min;

}

maze\_key=maze\_key+result[n/2][n/2];

***6.1.5. Encryption: (Using RC4 Algorithm)***

public class rc4\_decrypt {

private int[] S = new int[256];

private int[] T = new int[256];

private int keylen;

public int[] encrypt(int[] plaintext,int[] key)throws Exception{

if (key.length < 1 || key.length > 256) {

throw new Exception("key must be between 1 and 256 bytes");

}

else {

keylen = key.length;

for (int i = 0; i < 256; i++) {

S[i] = i;T[i] = key[i % keylen];

}

int j = 0;

for (int i = 0; i < 256; i++) {

j = (j + S[i] + T[i]) % 256;

S[i] ^= S[j];

S[j] ^= S[i];

S[i] ^= S[j];

}

}

int[] ciphertext = new int[plaintext.length];

int i = 0, j = 0, k, t;

for (int counter = 0; counter < plaintext.length; counter++) {

i = (i + 1) % 256;

j = (j + S[i]) % 256;

S[i] ^= S[j];

S[j] ^= S[i];

S[i] ^= S[j];

t = (S[i] + S[j]) % 256;

k = S[t];

ciphertext[counter] = plaintext[counter] ^ k;

}

return ciphertext;

}

public static void main(String[] args) throws Exception {

**//Magic Square Construction & Key Generation Part**

char ch[]=horizontal\_key.toCharArray();

int[] hor\_key = new int[horizontal\_key.length()];

for (int i = 0; i < horizontal\_key.length(); i++) {

hor\_key[i] = (int)ch[i];

}

char ch1[]=vertical\_key.toCharArray();

int[] ver\_key = new int[vertical\_key.length()];

for (int i = 0; i < vertical\_key.length(); i++) {

ver\_key[i] = (int)ch1[i];

}

System.out.println("\n\nEnter the Plain Text:");

String s = br.readLine();

char c1[]= s.toCharArray();

int[] text = new int[s.length()];

for (int i = 0; i < s.length(); i++) {

text[i] = (int)c1[i];

}

rc4 rc = new rc4();

int[] cipher = rc.encrypt(text,hor\_key); //encryption

FileWriter f=new FileWriter("Horizontal.txt");

String a="";

for(int i=0;i<cipher.length;i++){

a=Integer.toString(cipher[i]);

f.append(a);

f.append(" ");

}

f.close();

int[] cipher1 = rc.encrypt(text,ver\_key);

FileWriter f1=new FileWriter("Vertical.txt");

for(int i=0;i<cipher1.length;i++) {

a=Integer.toString(cipher1[i]);

f1.append(a);

f1.append(" ");

}f1.close();

System.out.print("\n\nCipher Text using Horizontal Key: ");

for (int i = 0; i < cipher.length; i++) {

System.out.println(cipher[i]);

}

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

System.out.print("\nCipher Text using Vertical Key: ");

for (int i = 0; i < cipher1.length; i++) {

System.out.println(cipher1[i]);

}

}

}

***6.1.6. Decryption:***

rc4\_decrypt rc = new rc4\_decrypt();

String a="";

File fi=new File("Horizontal.txt");

FileReader f=new FileReader(fi);

BufferedReader br1=new BufferedReader(f);

a=br1.readLine();

String []strs=a.split(" ");

int[] cipher=new int[(int)strs.length];

for(int i=0;i<strs.length;i++){

cipher[i]=Integer.parseInt(strs[i]);

}

f.close();

int[] decryption=rc.encrypt(cipher,hor\_key);

System.out.print("\n\n\n\nOriginalText using Horizontal Key: ");

for (int i = 0; i < decryption.length; i++){

System.out.print((char)decryption[i]); }

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

}

a="";

File fi1=new File("Vertical.txt");

FileReader f1=new FileReader(fi1);

BufferedReader br2=new BufferedReader(f1);

a=br2.readLine();

String []strs1=a.split(" ");

int[] cipher1=new int[(int)strs1.length];

for(int i=0;i<strs1.length;i++){

cipher1[i]=Integer.parseInt(strs1[i]);

}

f1.close();

int[] decryption1=rc.encrypt(cipher1,ver\_key);

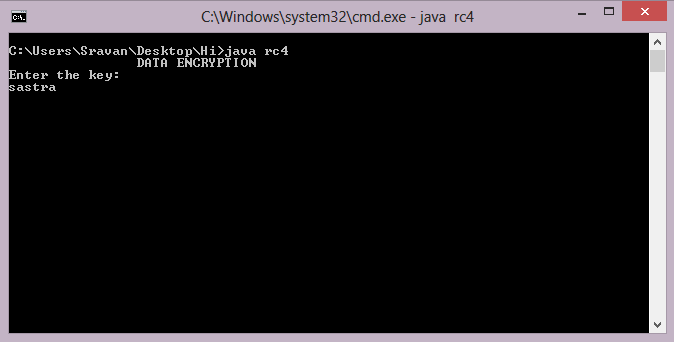
System.out.print("\n\n\n\nOriginal Text using Vertical Key: ");

for (int i = 0; i < decryption1.length; i++){

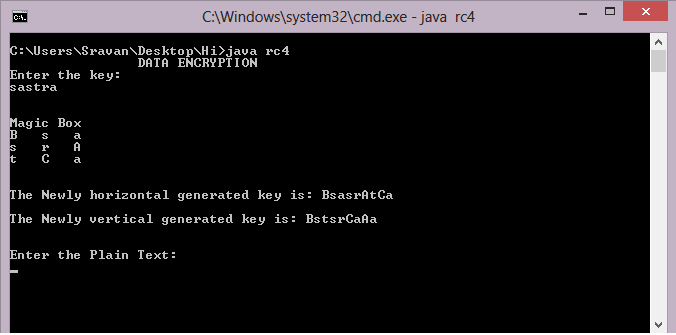
System.out.print((char)decryption1[i]);

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

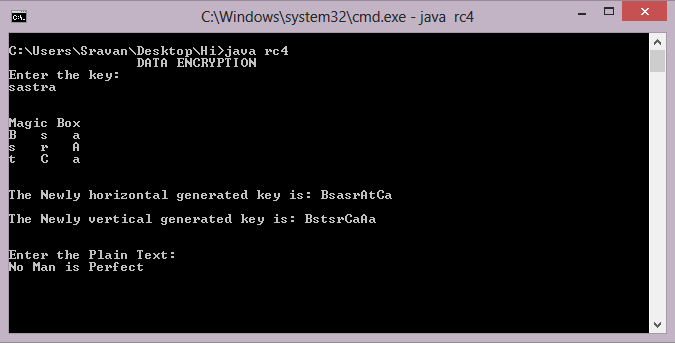
**6.2 Screenshots**

**

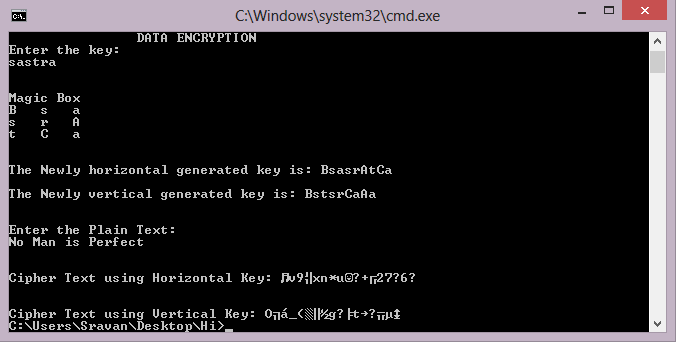
*Fig. no: 6.2.1 Getting the initial key from the user*

**

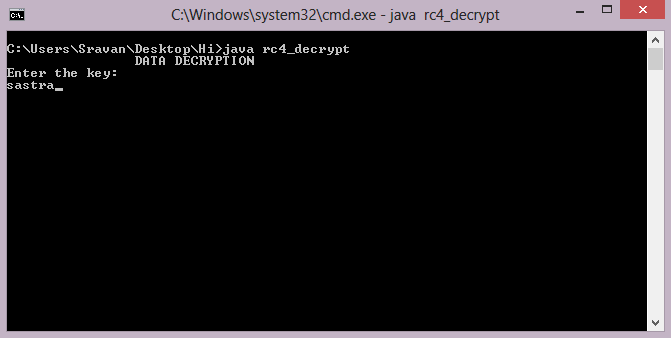
*Fig. no: 6.2.2 Magic Square Construction and Strengthened Generation*

**

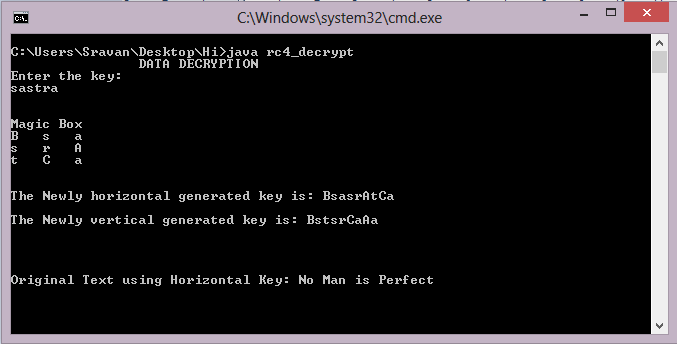
*Fig. no: 6.2.3 Getting the Plain Text from the User*

**

*Fig. no: 6.2.4 Generation of Cipher Text*

**

*Fig. no: 6.2.5 Getting the key from the user for Decryption*

**

*Fig. no: 6.2.6 Decryption of Original Message*

*IMPLEMENTATION*

**7. IMPLEMENTATION**

**7.1 Problems Faced**

The main objective is to provide purely random and dynamic keys for encryption. The process of traversing the magic square to extract different patterns of keys was difficult. The implementation of encryption algorithms using various keys simultaneously was tedious. Generating a highly secured and uncrackable cipher text was challenging.

**7.2 Lessons Learnt**

Through this project, we learnt in depth knowledge about cryptography and importance of secured communication given the advancement in hacking techniques. We learnt about the concepts of cryptanalysis and tried cracking out cipher text using brute force attack tools. We also learnt how important are the communications that happens between two parties nowadays.

*FUTURE SCOPE*

**8. FUTURE SCOPE**

In our proposed system, the order of Magic Square we use is ODD. This approach can also be implemented using Magic Squares of EVEN order. Here we use only certain patterns of key extraction. These patterns can be extended to further complex ways using several techniques computer graphics, where we can use pixel positions to extract the key from the magic square.

*REFERENCES*

**9. REFERENCES**

[1]G. Manikandan, R. Manikandan, G. Sundarganesh,2011, *A New approach for generating Strong Key in RivestCipher4 Algorithm*, Journal of Theoretical and Applied Information Technology,113-119.

[2]G.Manikandan, N.Sairam, M.Kamarasan, 2012,*A New Approach For Improving Data Security Using Iterative Blowfish Algorithm,*Research Journal of Applied Sciences, Engineering and Technology, 4(6) ,,603-607.

[3]G.Manikandan, N.Sairam, M.Kamarasan, A Hybrid Approach For Security Enhancement by Compressed Crypto-Stegno Scheme in Research Journal of Applied Sciences, Engineering and Technology,Vol 4(6) ,2012,

608-614.

[4] B.Karthikeyan, V.Vaithiyanathan, B. Thamotharan, M.Gomathymeenakshi and S.Sruthi, LSB Replacement Stegnography in an image Using Psudorandomised Key Generation. Research Journal of Applied Sciences, Engineering and Technology, 4(5), 2012, 491-494.

[5]S.Karthikeyan, N.Sairam, G.Manikandan, J.Sivaguru, A Parallel Approach For Improving Data Security in Journal of Theoretical and Applied Information Technology ,Vol 39(2) , May 2012,119-125.

[6]G.Manikandan, P.Rajendiran, K.Chakarapani, G.Krishnan,G. SundarGanesh, A Modified Crypto Scheme for Enhancing Data Security in Journal of Theoretical and applied information Technology ,Vol 35(2) , January 2012,149-154.

[7]S.Karthikeyan, N.Sairam, G.Manikandan, A New Approach For Enhancing Data Security Using Parallel Processing in Advances in Natural and Applied Sciences ,Vol 6(6) , July 2012,696-703.

[8]G.Manikandan,N.Sairam,M.Kamarasan, A New Approach for Secure Data Transfer based on Wavelet Transform in International Journal of Network Security, Vol 15(1) , 1-March 2013,88-94.