**Chapter 1**

**Introduction**

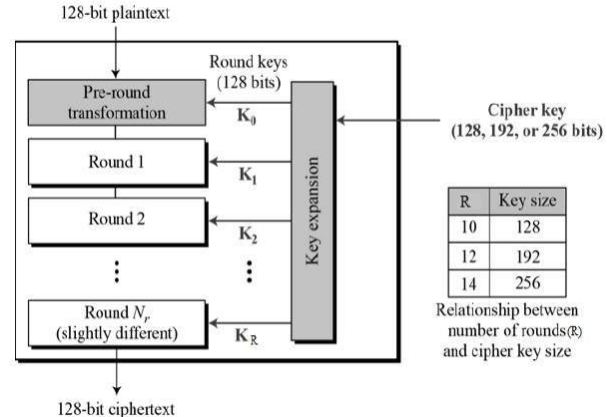
* 1. **Objective:** To implement AES (Advanced Encryption System) algorithm.
  2. **Introduction**

The project “Implementation of AES algorithm” will yield an application that will eradicate the crisis of finding a reliable blood donor. AES is an iterative rather than Feistel cipher. It is based on ‘substitution–permutation network’. It comprises of a series of linked operations, some of which involve replacing inputs by specific outputs (substitutions) and others involve shuffling bits around (permutations).

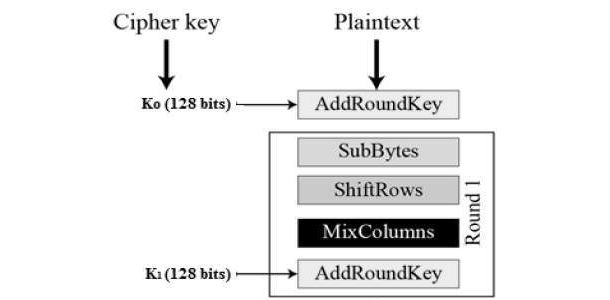
Interestingly, AES performs all its computations on bytes rather than bits. Hence, AES treats the 128 bits of a plaintext block as 16 bytes. These 16 bytes are arranged in four columns and four rows for processing as a matrix −

Unlike DES, the number of rounds in AES is variable and depends on the length of the key. AES uses 10 rounds for 128-bit keys, 12 rounds for 192-bit keys and 14 rounds for 256-bit keys. Each of these rounds uses a different 128-bit round key, which is calculated from the original AES key.

The schematic of AES structure is given in the following illustration

 **Figure 1.1: AES structure**

**The encryption process:**

**  
Figure 1.2: The encryption process**

**The decryption process:**

* Add round key
* Mix columns
* Shift rows
* Byte substitution

**1.3 Customer Requirement:**

Requirements stated by the client group are as below:

* Security issues
* Convenient i.e. Easy to use

**1.4 Specified Tools:**

**Language:**

* Java

**Environment:**

* Eclipse

**1.5 Problem Statement:**

In present day cryptography, AES is widely adopted and supported in both hardware and software. Till date, no practical cryptanalytic attacks against AES has been discovered. Additionally, AES has built-in flexibility of key length, which allows a degree of ‘future-proofing’ against progress in the ability to perform exhaustive key searches.

However, just as for DES, the AES security is assured only if it is correctly implemented and good key management is employed.

**1.6 Process Model:**

To eradicate the existing problem, the proposed system will be implemented in light of “Waterfall model”. It is also referred to as a linear-sequential life cycle model. It is very simple to understand and use. In a waterfall model, each phase must be completed fully before the next phase can begin. This type of software development model is basically used for the for the project which is small and there are no uncertain requirements. At the end of each phase, a review takes place to determine if the project is on the right path and whether or not to continue or discard the project. In this model software testing starts only after the development is complete. In waterfall model phases do not overlap.

**Figure 1.3**: Waterfall Process Model

**1.7 Why We Chose This Model:**

We choose Waterfall model because:

* The requirements to build the system are well defined and reasonably stable. This resembles with the Waterfall modeling.
* Moreover, the system will be developed in a systematic and sequential way which leads to a linear fashion. So, more reason to choose Waterfall model.

**Chapter 2**

**ER Diagram and Object Oriented Modeling**

**2.1 Objectives:**

* To describe the development process for implementation of AES algorithm.
* To evaluate functionality of the module of the system using UML Diagrams.

**2.2 Introduction:**

The Unified Modeling Language is a general-purpose, developmental, [modeling language](https://en.wikipedia.org/wiki/Modeling_language) in the field of [software engineering](https://en.wikipedia.org/wiki/Software_engineering) that is intended to provide a standard way to visualize the design of a system mostly known as UML. It’s an object oriented approach to process modeling.

Object-oriented approach offers a different model from the traditional process oriented approach, which is based on functions and procedures. In simplified terms, object-oriented systems development is a way to develop software by building self-contained modules or objects that can be easily replaced, modified, and reused. There are three different stages for implementing the process through these systems.

|  |  |  |
| --- | --- | --- |
| **Analysis** | **Design** | **Implementation** |
| Use Case Diagram  Class Diagram | State Diagram  Activity Diagram | System Implementation |

In order to describe the object oriented development process the whole system has been divided into three different subsystem and different UML diagrams have been used to describe each subsystem.

**2.3 Tools Used:**

* Paint
* Grapholite
* MS Word tools

**2.4 Use-Case Diagram:**

A diagram that depicts the interactions between the system, external systems and users is known as use-case diagram. It graphically describes who will use the system and in what ways the user expects to interact with the system.

**Use case** – a behaviorally related sequence of steps, both automated and manual, for the purpose of completing a single business task.

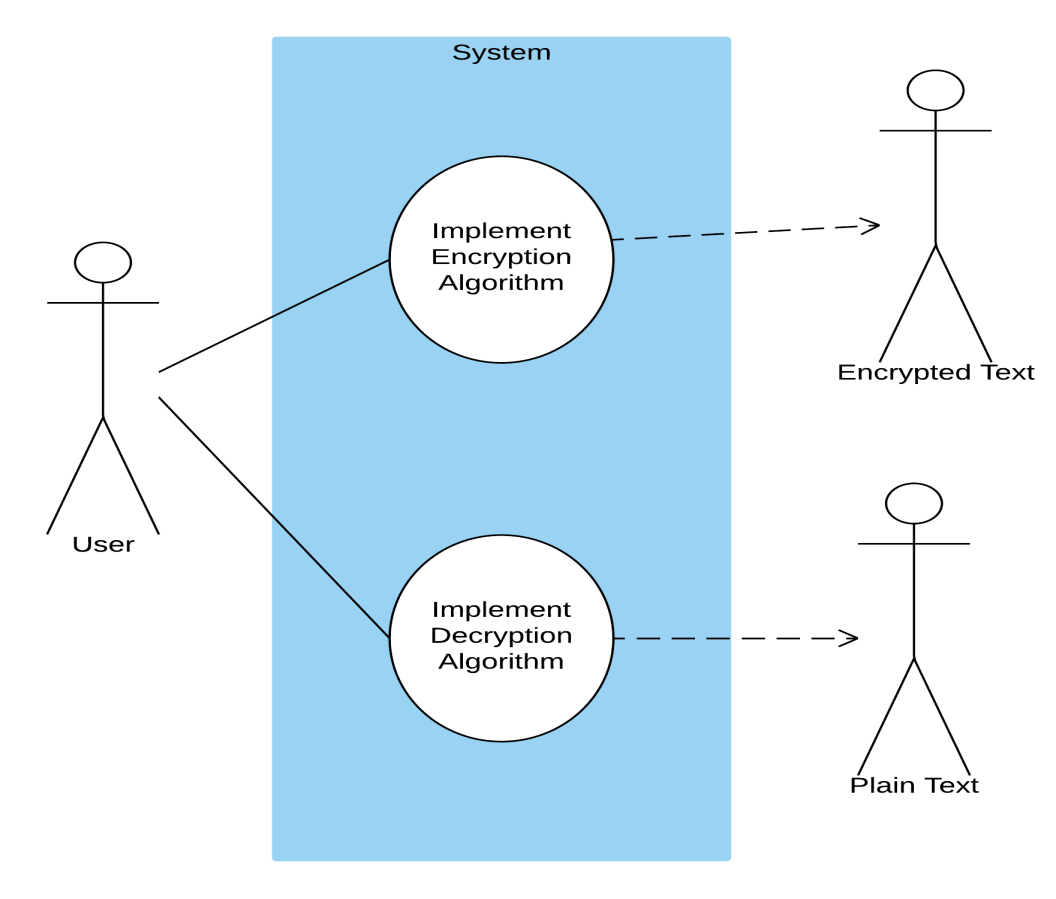
--Description of system functions from the perspective of external users in terminology they understand.

- Subset of the overall system functionality.

--Represented graphically by a horizontal ellipse with the name of the use case appearing above, below, or inside the ellipse.

**Actor** -- anything that needs to interact with the system to exchange information.

--Could be a human, an organization, another information system, an external device, or even time.

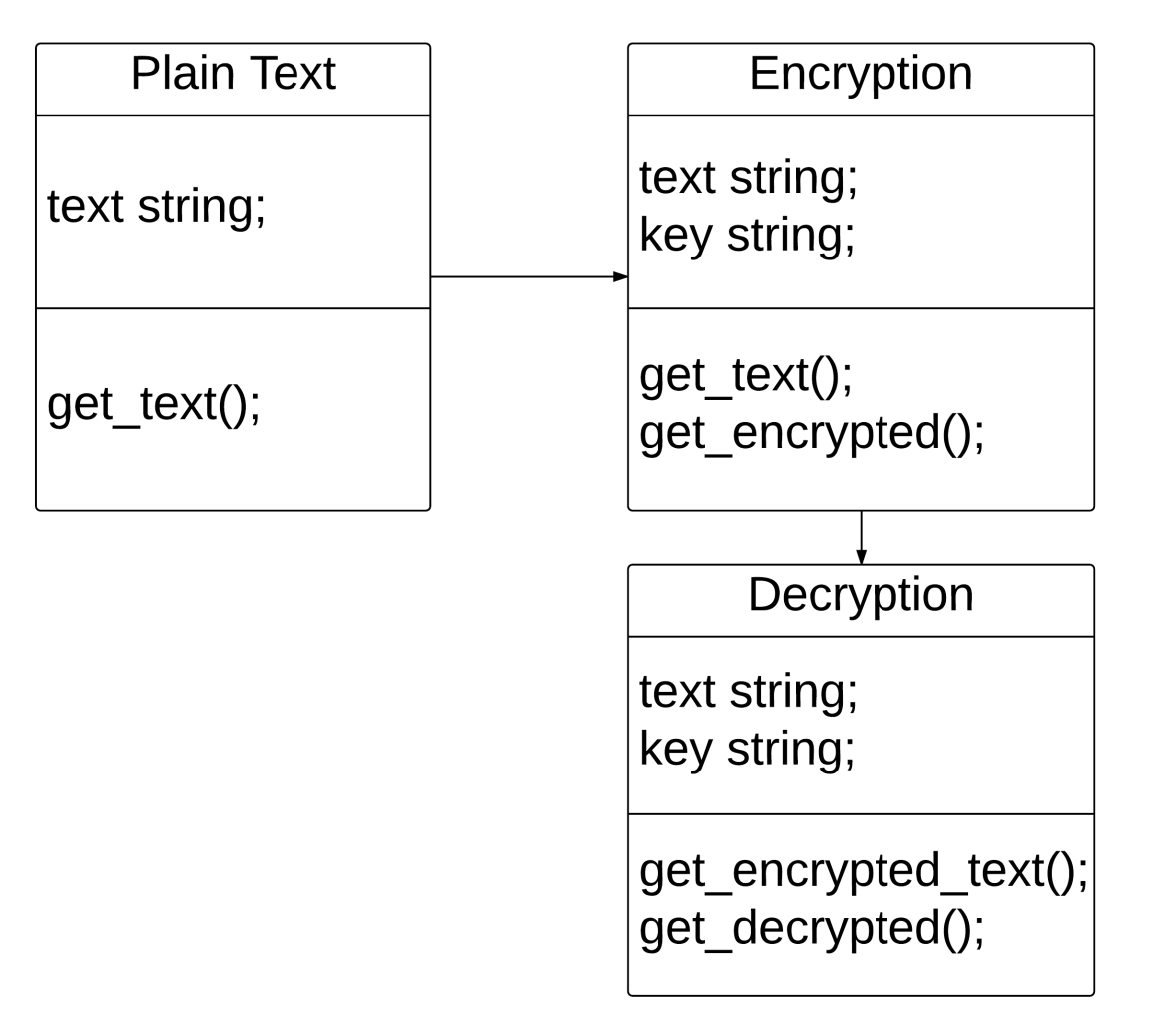


**Figure 2.1: Use-case diagram**

Above one is the use case diagram of our project. As it is not a full system that we are building so, it’s use case diagram is quite simple.

**2.5 Class Diagram:**

A class diagram in the unified modeling language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, and the relationships between the classes. It is represented using a rectangle with three compartments. Top compartment have the class name, middle compartments the attributes and the bottom compartment with operations.

****

**Figure 2.2: Class diagram**

Above one is the class diagram of our project. There are three classes in our project. One is named as ‘Plain text’. Others are ‘Encryption’ and ‘Decryption’.

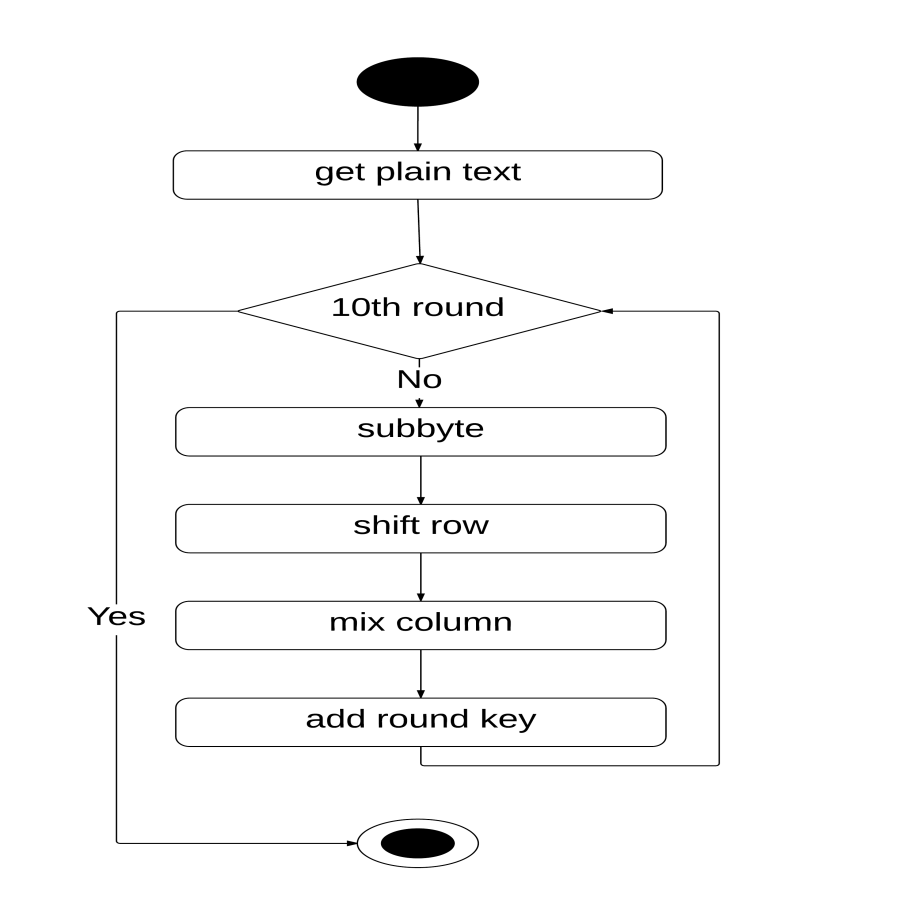
‘Plain Text’ class has a variable of data type string. And a method named get\_text().

‘Encryption’ class has two variables named ‘text’ and ‘key’ of string data type. And methods named ‘get\_text()’ and ‘get\_encrypted()’.

‘Decryption’ class has two variables named ‘text’ and ‘key’ of string data type. And methods named ‘get\_encrypted\_text()’ and ‘get\_decrypted()’.

### 2.6 Activity Diagram:

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control. An activity is shown as an rounded box containing the name of the operation.



**Figure 2.3: Activity diagram**

Above is the activity diagram of our project for doing encryption. It depicts the control flow of our project.

Through this flow, the given plaintext will go through under 10 rounds. Where, each round consists of 4 sub functions. Sub-byte, shift row, mix column and adding round key.

In Sub-byte sub-function, for every letter of the plain text a substituted letter will be obtained.

In shift row sub-function, the rows will be shifted as per algorithm.

In mix column sub-function, the columns of the matrix will be mixed for a higher security concern.

In add round key sub-function, the cipher key will be added with the so far processed data.

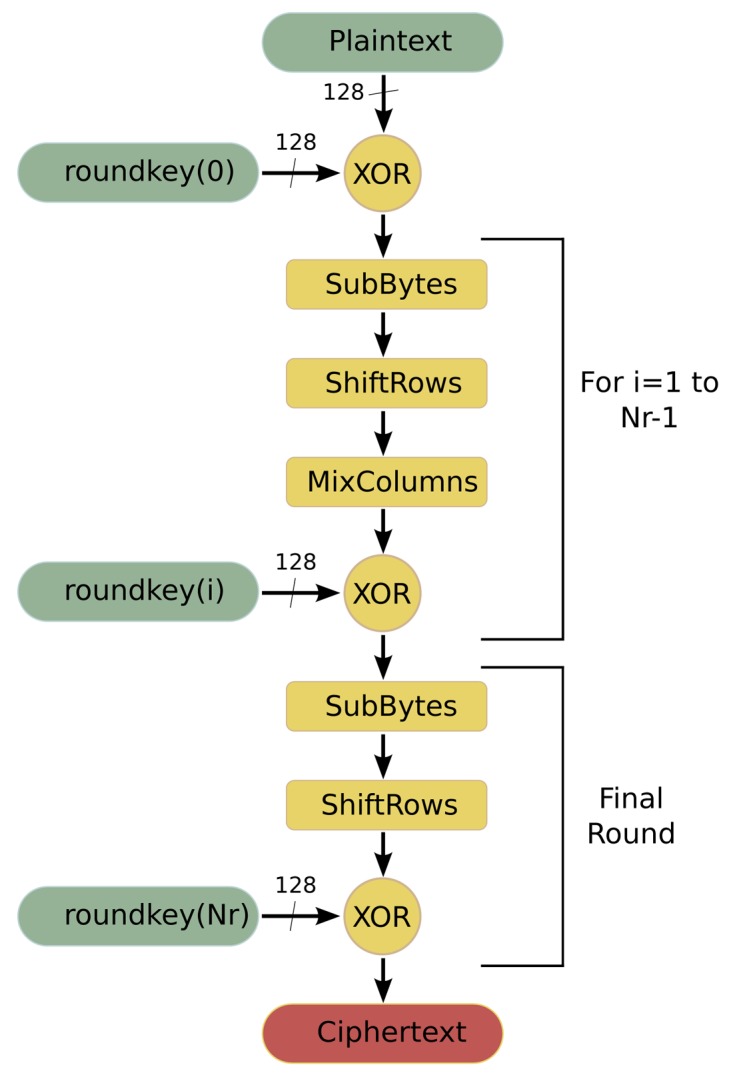
And as we will be implementing AES for 128 bit length key so, there will be 10 rounds. The first 9 rounds will be identical to each other. But the 10th round will be different than the others. And after 10th round the process will come to an end.

### 2.7 Flow Chart:

A diagram of the sequence of movements or actions of people or things involved in a complex system or activity is known as the flow chart.

There are 4 types of flow charts:

* Document flowcharts, showing controls over a document-flow through a system
* Data flowcharts, showing controls over a data-flow in a system
* System flowcharts, showing controls at a physical or resource level
* Program flowchart, showing the controls in a program within a system



**Figure 2.4: Flow chart**

The above one is the program flow chart of our project. It depicts how a plain text is converted into an encrypted text and vice versa.

**Chapter 3**

**Testing**

**3.1 Introduction:**

Software testing is an activity to check whether the actual results match the expected results and to ensure that the software system is defect free. It involves execution of a software component or system component to evaluate one or more properties of interest.

Software testing also helps to identify errors, gaps or missing requirements in contrary to the actual requirements. It can be either done manually or using automated tools. Some prefer saying Software testing as a white box and black box testing.

An early start to testing reduces the cost and time to rework and produce error-free software that is delivered to the client. However in Software Development Life Cycle, testing can be started from the Requirements Gathering phase and continued till the deployment of the software. It also depends on the development model that is being used. For example, in the Waterfall model, formal testing is conducted in the testing phase; but in the incremental model, testing is performed at the end of every increment/iteration and the whole application is tested at the end.

**3.2 Objective of testing:**

1. Finding defects which may get created by the programmer while developing the software.
2. Gaining confidence in and providing information about the level of quality to prevent defects.

**3.3 Classification of testing:**

1. White Box Testing

2. Black Box Testing.

**3.4 Black Box Testing:**

Black-box testing is a method of software testing that examines the functionality of an application based on the specifications. It is also known as Specifications based testing. Independent Testing Team usually performs this type of testing during the software testing life cycle.

This method of test can be applied to each and every level of software testing such as unit, integration, system and acceptance testing.

This method is named so because the software program, in the eyes of the tester, is like a black box; inside which one cannot see. This method attempts to find errors in the following categories:

* Incorrect or missing functions
* Interface errors
* Errors in data structures or external database access
* Behavior or performance errors
* Initialization and termination errors

**3.5 White Box testing:**

White-box testing (also known as clear box testing, glass box testing, transparent box testing, and structural testing) looks inside the software that is being tested and uses that knowledge as part of the testing process. If, for example, exception is thrown under certain conditions, test might want to reproduce those conditions. White-box testing requires internal knowledge of the system and programming skills. It provides internalperspective of the software under test.

**3.6 Basis Path Testing (BPT):**

Basis path testing is a white box method for designing test cases. The method analyzes the control flow graph of a program to find a set of linearly independent paths of execution. The method normally uses McCabe' cyclomatic complexity to determine the number of linearly independent paths and then generates test cases for each path thus obtained.

BPT uses a simple notation for the representation of control flow, called flow graph. Some of the basic notations for flow graph are described below:

* A circle in a graph represents a node, which stands for a sequence of one or more procedural statements
* A node containing a simple conditional expression is referred to as a predicate node
* Each compound condition in a conditional expression containing one or more Boolean operators (e.g., and, or) is represented by a separate predicate node
* A predicate node has two edges leading out from it (True and False)
* An edge, or a link, is a an arrow representing flow of control in a specific direction
* An edge must start and terminate at a node
* An edge does not intersect or cross over another edge
* Areas bounded by a set of edges and nodes are called regions.
* When counting regions, include the area outside the graph as a region, too.

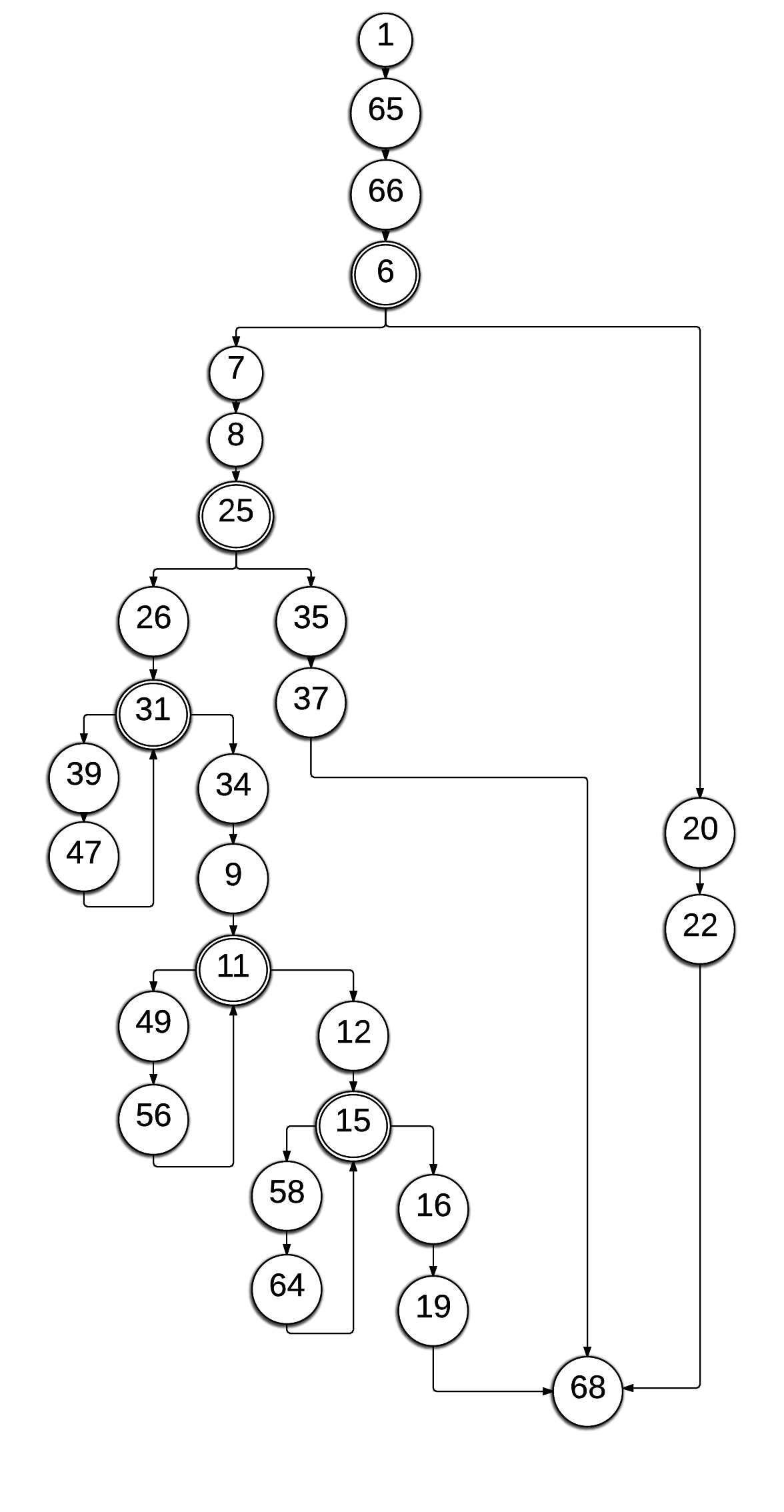
**3.7 Source Code:**

1. **public class** AES {
2. **byte**[] **skey** = **new byte**[1000];
3. String **skeyString**;
4. **static byte**[] *raw*;
5. String **inputMessage**,**encryptedData**,**decryptedMessage**;
6. **public** AES() {
7. **try** {
8. generateSymmetricKey();
9. **inputMessage**=JOptionPane.showInputDialog(**null**,**"Enter message to  
    encrypt"**);
10. **byte**[] ibyte = **inputMessage**.getBytes();
11. **byte**[] ebyte=*encrypt*(*raw*, ibyte);
12. String encryptedData = **new** String(ebyte);
13. System.***out***.println(**"Encrypted message "**+encryptedData);
14. JOptionPane.showMessageDialog(**null**,**"Encrypted  
     Data"**+**"\n"**+encryptedData);
15. **byte**[] dbyte= *decrypt*(*raw*,ebyte);
16. String decryptedMessage = **new** String(dbyte
17. System.***out***.println(**"Decrypted message** +decryptedMessage
18. JOptionPane.showMessageDialog(**null**,**"Decrypted Data  
     "**+**"\n"**+decryptedMessage);
19. }
20. **catch**(Exception e) {
21. System.***out***.println(e);
22. }
23. }
24. *//Symmetric key generation*
25. **void** generateSymmetricKey() {
26. **try** {
27. Random r = **new** Random();
28. **int** num = r.nextInt(10000);
29. String knum = String.*valueOf*(num);
30. **byte**[] knumb = knum.getBytes();
31. **skey**=*getRawKey*(knumb);
32. **skeyString** = **new** String(**skey**);
33. System.***out***.println(**"AES Symmetric key = "**+**skeyString**);
34. }
35. **catch**(Exception e) {
36. System.***out***.println(e);
37. }
38. }
39. **private static byte**[] getRawKey(**byte**[] seed) **throws** Exception {
40. KeyGenerator kgen = KeyGenerator.*getInstance*(**"AES"**);
41. SecureRandom sr = SecureRandom.*getInstance*(**"SHA1PRNG"**);
42. sr.setSeed(seed);
43. kgen.init(128, sr);
44. SecretKey skey = kgen.generateKey();
45. *raw* = skey.getEncoded();
46. **return** *raw*;
47. }
48. *//encrypt portion*
49. **private static byte**[] encrypt(**byte**[] raw, **byte**[] clear) **throws** Exception
50. {
51. SecretKeySpec skeySpec = **new** SecretKeySpec(raw, **"AES"**);
52. Cipher cipher = Cipher.*getInstance*(**"AES"**);
53. cipher.init(Cipher.***ENCRYPT\_MODE***, skeySpec);
54. **byte**[] encrypted = cipher.doFinal(clear);
55. **return** encrypted;
56. }
57. *//decrypt portion*
58. **private static byte**[] decrypt(**byte**[] raw, **byte**[] encrypted) **throws** Exception {
59. SecretKeySpec skeySpec = **new** SecretKeySpec(raw, **"AES"**);
60. Cipher cipher = Cipher.*getInstance*(**"AES"**);
61. cipher.init(Cipher.***DECRYPT\_MODE***, skeySpec);
62. **byte**[] decrypted = cipher.doFinal(encrypted);
63. **return** decrypted;
64. }
65. **public static void** main(String args[]) {
66. AES aes = **new** AES();
67. }
68. }

**3.8 Flow Graph of corresponding code:**

Flow Graph is defined as a function in a program that can be represented as a control flow graph and the nodes in the flow graph are defined as program statements while the directed edges are the flow of control. A Flow Graph consists of nodes and edges. The two nodes in the Flow Graph can be either unconnected or connected by an edge in either direction.

While tracing a path from a source to a sink a back edge is an edge that leads back to a node that has already been visited. The Flow Graph contains one source node and one sink.   
A source node is the node that has no incoming edges while a sink node is the node with no outgoing edges. A program's function may contain more than one sink node, but this graph can be converted into a graph with only one sink. There are some languages that allow more than one source. This construct is very rare and not used in Structured Programming.



**Figure 3.1: Flow Graph of the Source Code**

**3.9 Calculating Cyclomatic Complexity:**

No. of Nodes=27

No. of Edges=31

Cyclometic complexity, V(G)=E-N+2 = 31-27+2=4+2=6

**3.10 Basis set of independent paths:**

Path 1: 1-65-66-6-7-8-25-26-31-34-9-11-12-15-16-19-68

Path 2: 1-65-66-6-7-8-25-26-31-39-47-31-34-9-11-12-15-16-19-68

Path 3: 1-65-66-6-7-8-25-26-31-34-9-11-49-56-11-12-15-16-19-68

Path 4: 1-65-66-6-7-8-25-26-31-34-9-11-12-15-58-64-16-19-68

Path 5: 1-65-66-6-7-8-25-35-37-68

Path 6: 1-65-66-6-20-22-68

**Chapter 4**

**Implementation**

In this chapter full flow of our software with user interface and instructions are provided. At the end final conclusion of our project and future recommendations are stated.

# **4.1 User interface**

**Final Conclusion and Future Recommendation**