



Jaypee University of Information Technology
Solan (H.P.)

LEARNING RESOURCE CENTER

Acc. Num. *SP04034* Call Num:

General Guidelines:

- ◆ Library books should be used with great care.
- ◆ Tearing, folding, cutting of library books or making any marks on them is not permitted and shall lead to disciplinary action.
- ◆ Any defect noticed at the time of borrowing books must be brought to the library staff immediately. Otherwise the borrower may be required to replace the book by a new copy.
- ◆ The loss of LRC book(s) must be immediately brought to the notice of the Librarian in writing.

Learning Resource Centre-JUIT



SP04034

ENCRYPTION AND DECRYPTION OF FILES

CERTIFICATE

**Submitted in partial fulfillment of the Degree of Bachelor of
Technology**

By

Kanishk Kumar-041097

Pankaj Triapthi-041010

(Mr. Anmol Vasudeva)
(Project Coordinator)
Associate Lecturer,
Department of Computer Science
and Information Technology,
Jaypee University of Information Technology,
Waknaghat, Solan (H.P.)



MAY-2008

**DEPARTMENT OF ELECTRONICS AND
COMMUNICATION ENGINEERING
JAYPEE UNIVERSITY OF INFORMATION
TECHNOLOGY-WAKNAGHAT**

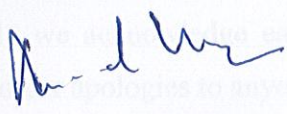
CERTIFICATE

ACKNOWLEDGEMENT

This is to certify that the work entitled, “**ENCRYPTION AND DECRYPTION OF FILES**” submitted by **Kanishk Kumar (041097)** and **Pankaj Triapthi (041010)** in partial fulfillment for the award of degree of Bachelor of Technology in Electronics and Communication Engineering of Jaypee University of Information Technology has been carried out under my supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree or diploma.

directions and helped us a lot for publishing research papers which had been a great source of motivation. We are greatly indebted to him for the support he has given to us and for his words of encouragement and motivation.

We also acknowledge many helpful comments and suggestions received from our teachers of the concerned department for improving and fulfillment of this project. Finally we acknowledge each and everyone who helped us directly or indirectly and extend our regards to anyone we have failed to mention.


[Mr. Amol Vasudeva]
(Project Coordinator)

Associate Lecturer,
Department of Computer Science
and Information Technology,
Jaypee University of Information Technology,
Waknaghat, Solan (H.P.)

Pankaj Triapthi

041010

Signature
Pankaj
Pankaj

ACKNOWLEDGEMENT

We wish to thank and acknowledge our project guide and our mentor for his great contribution in our project. He always guided us time to time and was always there to help us out any time we needed him for the project. Being a great mentor himself, he showed us the path for successful completion of project. Besides helping and guiding us in our project he also gave directions and helped us a lot for publishing research papers which had been a great source of motivation. We are greatly indebted to him for the support he has given to us and for his words of encouragement and motivation.

We also acknowledge many helpful comments and suggestions received from our teachers of the concerned department for improving and fulfillment of this project. Finally we acknowledge each and everyone who helped us directly or indirectly and extend our apologies to anyone we have failed to mention.

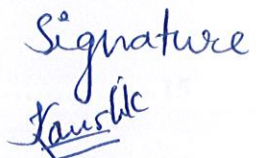
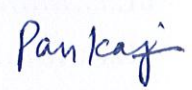
Name	Roll No.	Signature
Kanishk Kumar	041097	
Pankaj Teeripathi	041010	

Table of Contents

CHAPTER-I: Introduction.....	1
1.1 Project Motivation.....	1
1.2 Project Objective.....	1
1.3 System Development Life Cycle.....	2
1.3.1 Project Planning.....	3
1.3.2 Requirement Analysis.....	4
1.3.3 Design.....	6
1.3.4 Development.....	7
1.3.5 Integration & Testing.....	8
1.3.6 Installation & Acceptance.....	9
1.4 Generic Stages.....	10
1.4.1 Kickoff Process.....	11
1.4.2 Information Iteration Process.....	11
1.4.3 Formal Iteration Process.....	12
1.4.4 In-stage Assessment Process.....	13
1.4.5 Stage Exit Process.....	14
CHAPTER-II: Cryptography.....	15
2.1 What is Cryptography?	15
2.2 Purpose of Cryptography.....	15
2.3 Cryptographic Functions.....	16
2.3.1 Secret Key Cryptography.....	16
2.3.2 Public Key Cryptography.....	18
2.3.3 Hash Functions.....	18

2.4	Cryptographic Algorithms.....	19
2.4.1	Secret Key Algorithms.....	19
2.4.2	Public Key Algorithms.....	22
2.4.3	Hash Functions.....	24
2.5	Advance Encryption Standard.....	26
2.5.1	Algorithm Specification.....	26
2.5.2	Cipher.....	27
2.5.3	SubBytes() Transformation.....	28
2.5.4	ShiftRows() Transformation.....	30
2.5.5	MixColumns() Transformation.....	31
2.5.6	AddRoundKey() Transformation.....	32
2.5.7	Key Expansion.....	33
2.5.8	Inverse Cipher.....	35
2.5.9	InvShiftRows() Transformation.....	35
2.5.10	InvSubBytes() Transformation.....	36
2.5.11	InvMixColumns() Transformation.....	37
2.5.12	Inverse of the AddRoundKey() Transformation.....	38
2.5.13	Advantages.....	38
2.5.14	Disadvantages.....	39
CHAPTER-III:	Project Planning.....	40
3.1	Project Plan.....	41
3.2	Milestones and Deliverables.....	41
3.3	Project Scheduling.....	43
3.4	Bar Charts & Activity Networks.....	44

CHAPTER-VIII: Installation & Acceptance.....	106
8.1 Quality Management.....	106
8.2 Quality Assurance & standards.....	107
8.3 Product & Process Standards	108
8.4 Quality Planning.....	109
CHAPTER-IX: Conclusion and Future Work.....	110
CHAPTER-X: Installation Guide.....	111
10.1 S/W and H/W requirements.....	111
10.2 Installation Procedure.....	112
10.3 Compilation and Execution.....	113
10.4 Test Layouts and Expected Output.....	113
BIBLIOGRAPHY.....	116

List of Figures:	Page
System Development Life Cycle.....	2
Project Planning.....	3
Requirements Analysis.....	5
Design Stage.....	6
Development Stage.....	7
Integration and Testing.....	8
Integration & Acceptance.....	10
Generic Stage.....	11
Algorithm Specification.....	27
Byte Substitution.....	29
Shift Row Transformation.....	31
Mix Colum Transformation.....	32
AddRoundKey Transformation	33
InvShiftRows Transformation.....	36
InvMixColumns Transformation.....	37
Project Scheduling.....	43
Different Readers.....	46
Statement of Purpose.....	52
Cartesian Hierarchy.....	53
Data Flow Diagram for File Encryption.....	54
Data Flow Diagram for File Decryption.....	55
Black Box Testing for Encryption Process.....	101
Building Test Case.....	102
Black Box Testing for Decryption Process.....	103
Building Test Case.....	104

List of Abbreviations:

1. AES :	Advance Encryption Standard
2. CBC :	Cipher Block Chaining
3. CFB :	Cipher Feedback
4. DES :	Data Encryption Standards
5. DSA :	Digital Signature Algorithm
6. ECB :	Electronic Code Book
7. ECC :	Electric Curve Cryptography
8. IDEA :	International Data Encryption Algorithm
9. KISA :	Korea Information Security Agency
10. KEA :	Key Exchange Algorithm
11. MEC :	Mitsubishi Electric Corporation
12. MD :	Message Digest Algorithm
13. NDT :	Nippon Telegraph & Telephone
14. OFD :	Output feedback
15. PKC :	Public Key Cryptography
16. PKCS :	Public Key Cryptography Standard
17. QA :	Quality Assurance
18. QAR :	Quality Assurance Reviewer
19. RTM :	Requirement Traceability
20. SAER :	Secure and Fast Encryption Routing
21. SDLC :	System Development Life Cycle
22. SOP :	Statement of Purpose
23. SHA :	Secure Hash Algorithm
24. UMTS : System	Universal Mobile Telecommunication

ABSTRACT

During this time when the internet provides essential communication between tens of millions of people and is being increasingly used as a tool for commerce, security becomes a tremendously an important issue to deal with. The expansion of the connectivity of computers makes ways of protecting data and messages from tampering or unauthorized reading. It is thus up to the user to ensure that communications which are expected to remain private actually do so. There are many aspects to security and many applications, ranging from secure commerce and payments to private communications and protecting passwords. One essential aspect of secure communication is that of cryptography, which the focus of this project is.

This project aims at designing and developing a graphical user interface which can encrypt and decrypt a file selected by the user. The file can be text, audio, video or picture type. It thus allows security to user as well as his/her system from unauthorized users as well as malicious sources.

1.2 Project objective

The primitive objectives of the project thought are the following:

- To propose a system which can encrypt and decrypt selected files on a system using standard encryption and decryption algorithm to protect authorized users against unauthorized users.

CHAPTER-I

INTRODUCTION

1.1 Project Motivation

If the confidentiality or accuracy of your information is of any value at all, it should be protected to an appropriate level and if the unauthorized disclosure or alteration of the information could result in any negative impact, it should be secured.

Over the time cryptography has become an important concern for programmers and users all around, during this time when the internet provides essential communication between tens and millions of people and is being increasingly used as a tool for commerce ,security become a tremendously important issue to deal with.

There are many aspects to security and many applications, ranging from secure commerce and payments to private communications and protecting passwords. one essential aspect for secure communications is that of cryptography, which is the focus of this project.

The desire of a file security program that can password protect, lock, hide and encrypt any no of files inspired us to design and develop this project.

1.2 Project objective

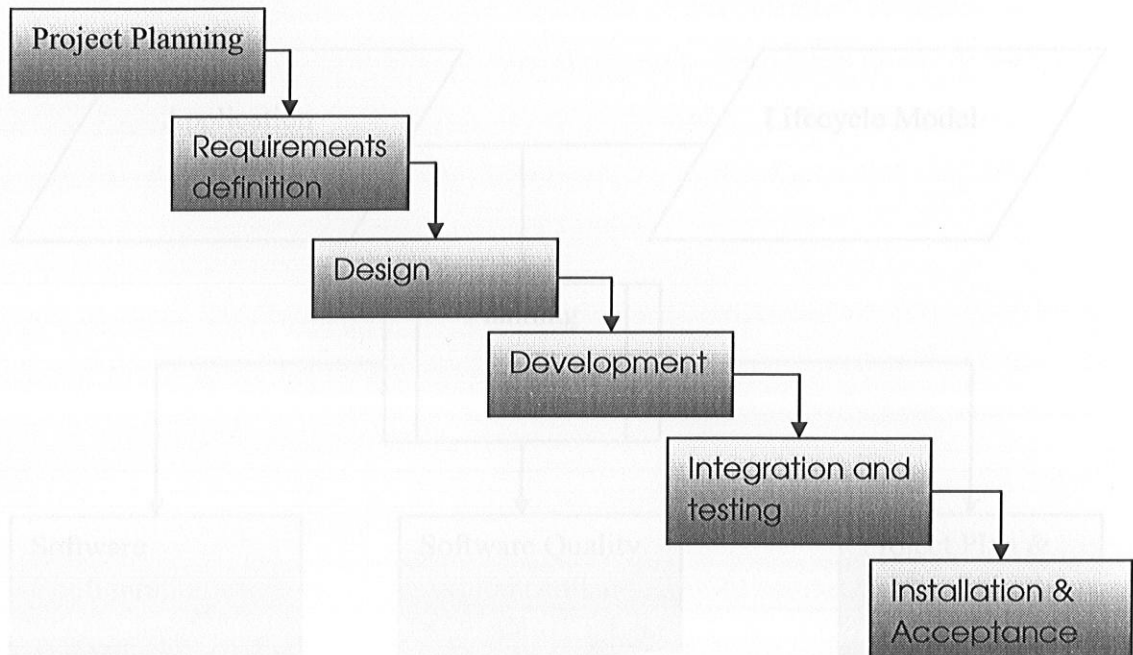
The primitive objectives of the project thought are the following:

- To propose a system which can encrypt and decrypt selected files on a system using standard encryption and decryption algorithm to protect authorized users against unauthorized users.

1.3 System Development Life Cycle

The system development life cycle followed by us in the project life cycle encompasses the following phases:

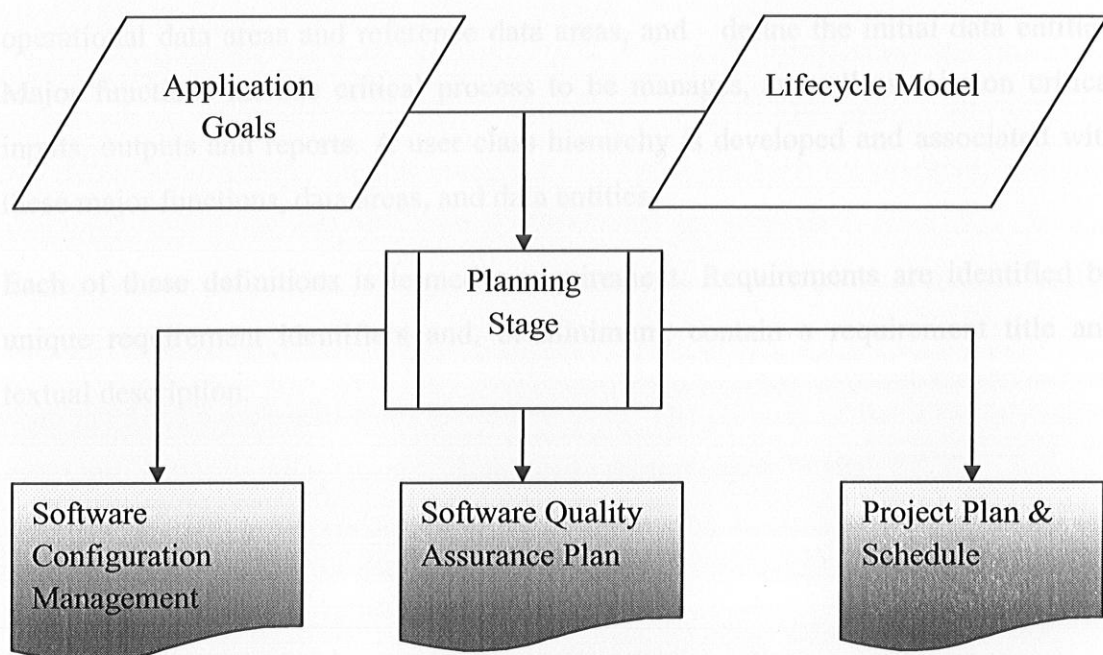
- Project planning
- Requirements definition
- Design
- Development
- Integration and testing
- Installation and acceptance



1.3.1 Project Planning

The six stages of the SDLC are designed to build on one another, taking the outputs from previous stage, adding additional efforts, and producing results that leverage the previous effort and are directly traceable to the previous stages. This top down approach is intended to result in quality product that satisfies the original intentions of the customer.

The planning stage stabilizes a bird's eye view of the intended software product, and uses this to establish the basic project structure, evaluate feasibility and risks associated with the project, and describe appropriate management and technical approaches.



The most critical session of the project plan is a listing of high level product requirements, also referred to as goals. All of the software product requirements to be developed during the requirements definition stage flow from one or more of these

goals. The minimum information for each goal consists of a title and textual description, although additional information and references to external documents may be included.

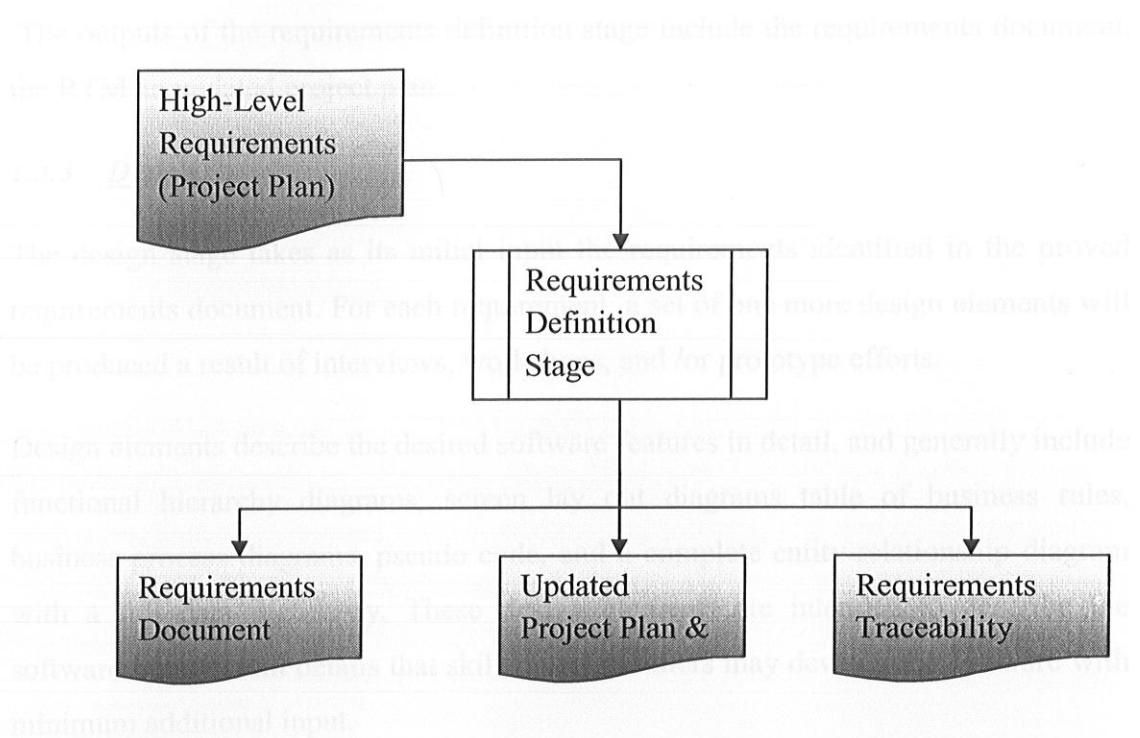
The outputs of the project planning stage are the configuration management plan, the quality assurance Plan, and the project plan and schedule, with a detailed listing of scheduled activities for the upcoming requirements stage, and high level estimates of effort for the out stages.

1.3.2 Requirements Analysis

The requirements gathering process takes as its input the goals identified in the high level requirements section of the project plan. Each goal will be refined into a set of one or more requirements.

These requirements define the major functions of the intended application, define operational data areas and reference data areas, and define the initial data entities. Major functions include critical process to be manages, as well as mission critical inputs, outputs and reports. A user class hierarchy is developed and associated with these major functions, data areas, and data entities.

Each of these definitions is termed a requirement. Requirements are identified by unique requirement identifiers and, at minimum, contain a requirement title and textual description.



These requirements are fully described in the primary deliverables for this stage: requirements document and the requirements traceability matrix (RTM). The requirement document contains complete description of each requirement, including diagrams and references to external documents as necessary. Note that detailed listings of data base tables and fields are not included in the requirements document.

The title of each requirement is also placed into the first version of the RTM, along with the title of each goal from the project plan. The purpose of the RTM is to show that the product components developed during each stage of the software development life cycle are formally connected to the components developed in the prior stages.

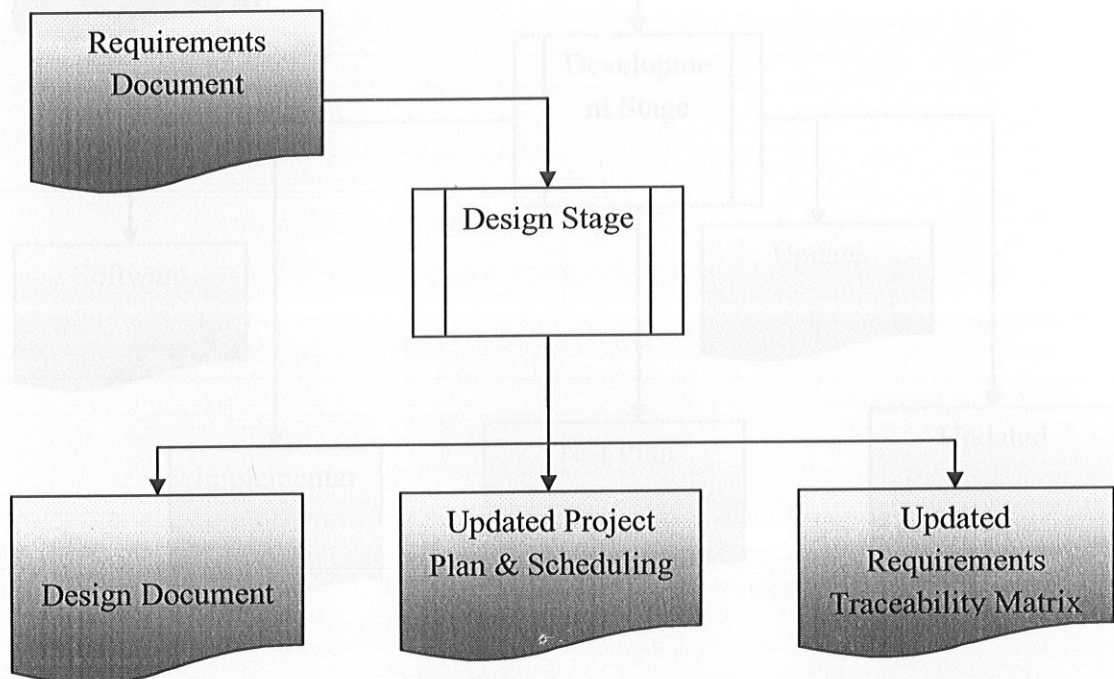
In the requirements stage, the RTM consists of a list of high level requirements, or goals, by title, with the listing of associated requirements for each goal, listed by requirement title. In this hierarchical listing the RTM shows that each requirement developed during this stage is formally linked to a specific product goal. In this format, each requirement can be traced to a specific product goal, hence the term requirements traceability.

The outputs of the requirements definition stage include the requirements document, the RTM an updated project plan.

1.3.3 Design

The design stage takes as its initial input the requirements identified in the proved requirements document. For each requirement, a set of one more design elements will be produced a result of interviews, workshops, and /or prototype efforts.

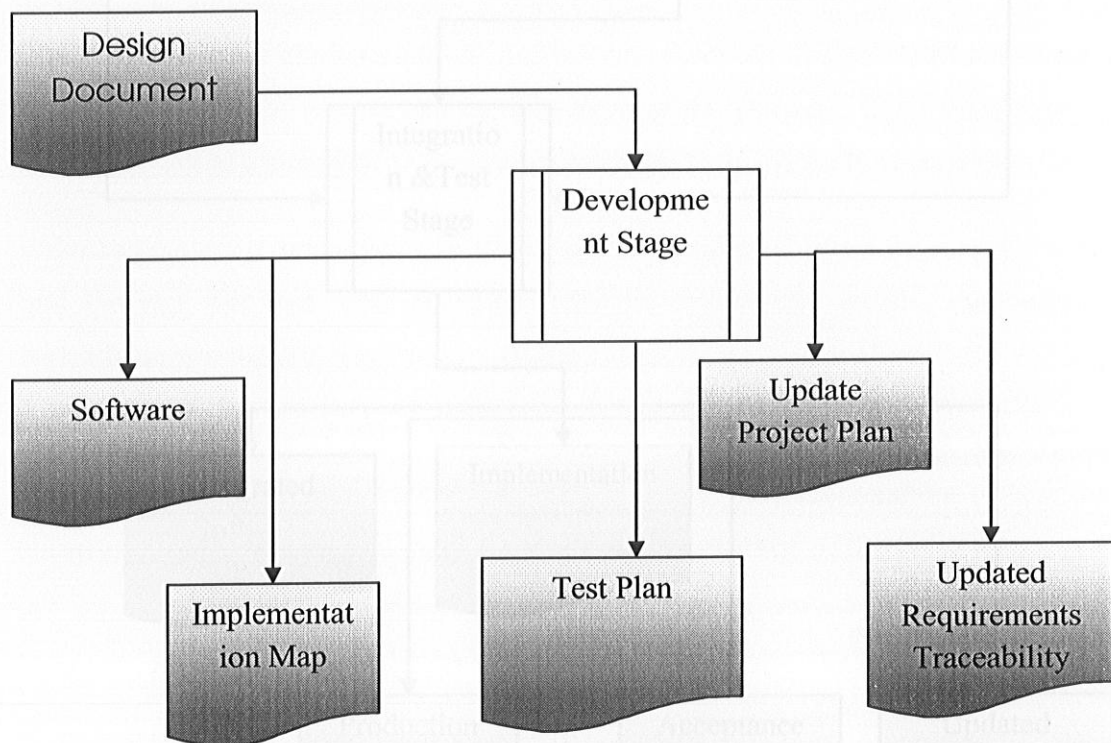
Design elements describe the desired software features in detail, and generally include functional hierarchy diagrams; screen lay out diagrams table of business rules, business process diagrams, pseudo code, and a complete entity-relationship diagram with a full data dictionary. These design elements are intended to describe the software in sufficient details that skilled programmers may develop the software with minimum additional input.



When the design document is finalized and accepted, the RTM is updated to show that each design element is formally associated with a specific requirement. The outputs of the design stage are the design document, an updated RTM, and an updated project plan.

1.3.4 Development

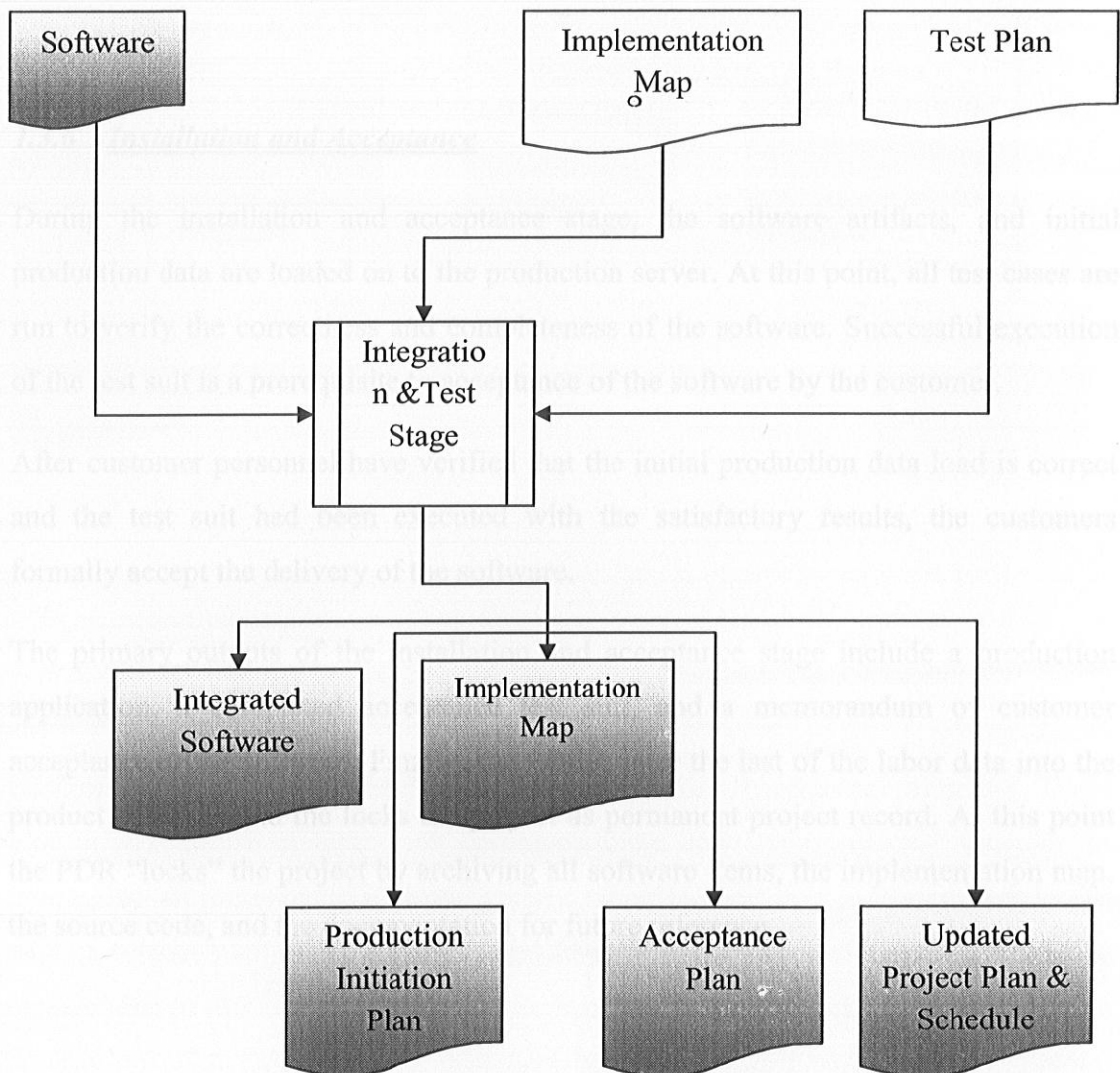
The development stage takes as its primary input the design element described in the approved design document. For each design document a set of one or more software artifacts will be produced. Software artifacts include but are not limited to menus, dialogues, data management form data reporting formats, and specialized procedures and functions. Appropriate test cases will be developed for each set of functionally related software artifacts.



The outputs of the development stage include a fully functional set of software that satisfies the requirements and design elements previously documented, an implementation map that identifies the primary code entry points for major system functions, a test plan that describes the test cases to be used to validate the correctness and completeness of the software, an updated RTM, and an updated project plan.

1.3.5 Integration and Testing

During the integration and test stage, software artifacts, and test data are migrated from the development environments to a separate test environments. At this point all the test cases are run to verify the correctness and completeness of the software. Successful execution of the test suit confirms a robust and complete migration capability,



During this stage, reference data is finalized for production use and production users are identified and linked to their appropriate roles. The final reference data (or links to reference data source files) and production user list are completed into the production initiation plan.

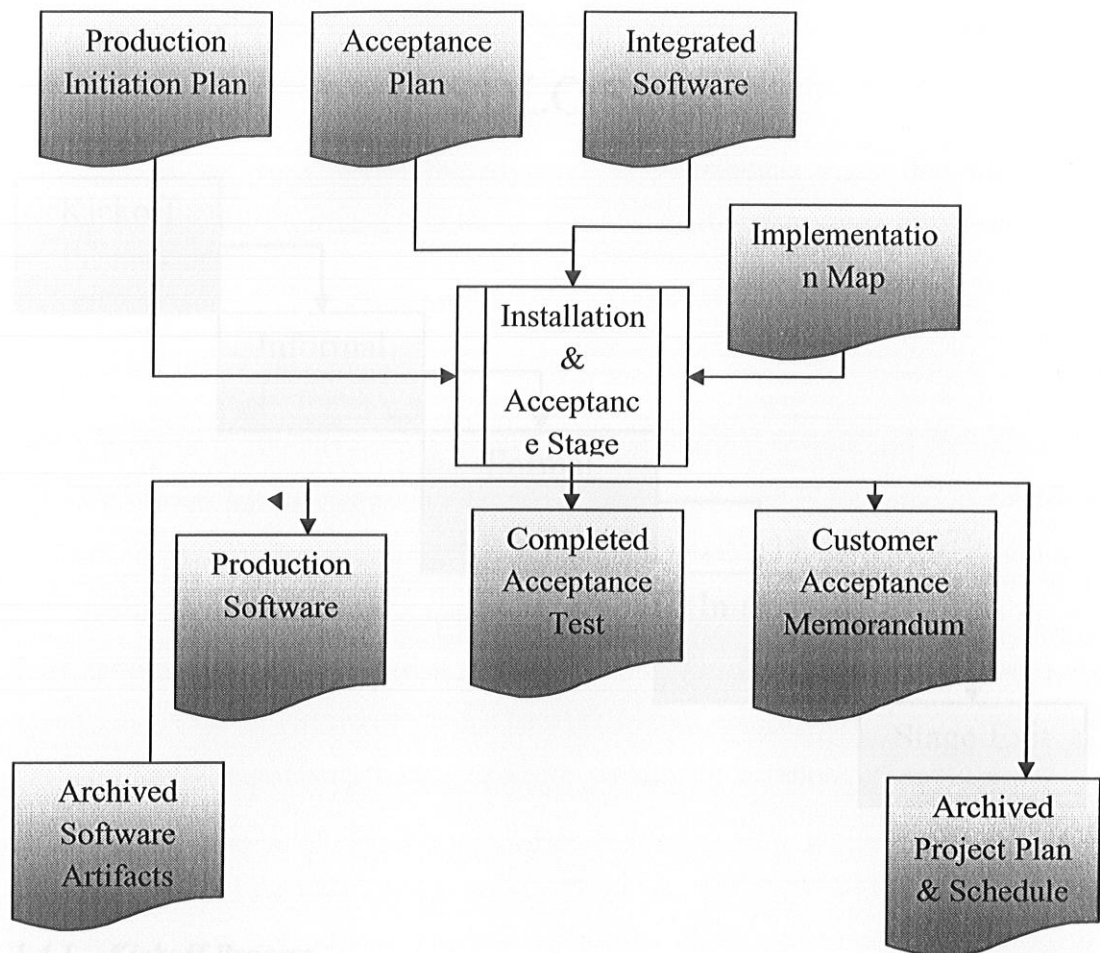
The outputs of the integration and test stage include integrated software, an implementation map, a production initiation plan that describes a reference data and production users, an acceptance plan which contains the final suit of test cases, and an updated project plan.

1.3.6 Installation and Acceptance

During the installation and acceptance stage, the software artifacts, and initial production data are loaded on to the production server. At this point, all test cases are run to verify the correctness and completeness of the software. Successful execution of the test suit is a prerequisite to acceptance of the software by the customer.

After customer personnel have verified that the initial production data load is correct and the test suit had been executed with the satisfactory results, the customers formally accept the delivery of the software.

The primary outputs of the installation and acceptance stage include a production application, a completed acceptance test suit, and a memorandum of customer acceptance of the software. Finally, the PDER enter the last of the labor data into the product schedule and the locks the project as permanent project record. At this point the PDR "locks" the project by archiving all software items, the implementation map, the source code, and the documentation for future reference.



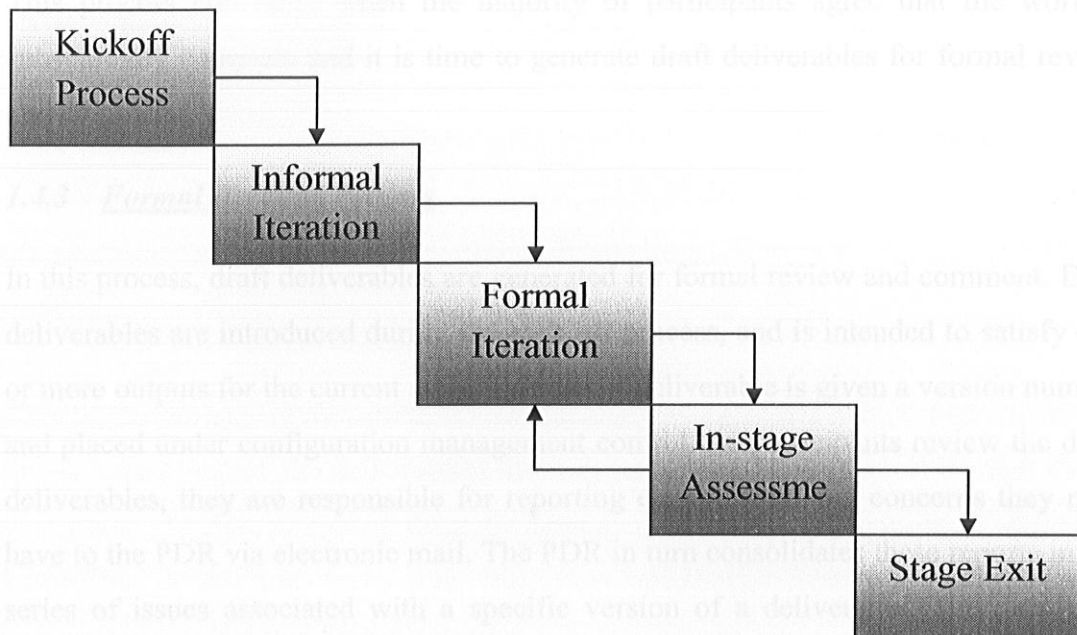
1.4.1 Kickoff Process

Each stage is initiated by a kickoff meeting, which can be conducted either in person, or by Web teleconferencing. The purpose of the kickoff meeting is to review the output of the previous stage, go over any additional inputs required by that

1.4 Generic Stages

Each of the stages of the development lifecycle follows five internal processes. These processes establish a pattern of communication and documentation intended to familiarize all participants with the current situation, and thus minimize risk to the current project plan. This generic stage description is provided to avoid repetitive descriptions of this internal process in each of the following software lifecycle stage description. The five standard processes are Kickoff, Informal iteration, Formal iteration, in stage assessment, and Stage exit.

SDLC Stage



1.4.1 Kickoff Process

Each stage is initiated by a kickoff meeting, which can be conducted either in person, or by Web teleconferences. The purpose of the kickoff meeting is to review the output of the previous stage, go over any additional inputs required by that particular stage, and examine the anticipated activities and required outputs of the current stage. Review the current project schedule, and review any open issues. The PDR is responsible for preparing the agenda and materials to be presented at this meeting. All project participants are invited to attend the kickoff meeting for each stage.

1.4.2 Information Iteration Process

Most of the creative work for a stage occurs here. Participants work together to gather additional information and refine stage inputs into draft deliverables. Activities of this stage may include interviews, meetings, the generation of prototypes, and electronic correspondence. All of these communications are deemed informal, and or not recorded as minutes, documents of record, controlled software, or official

memoranda. The intent here is to encourage, rather than inhibit the communication process.

This process concludes when the majority of participants agree that the work is substantially complete and it is time to generate draft deliverables for formal review and comment.

1.4.3 Formal Iteration Process

In this process, draft deliverables are generated for formal review and comment. Each deliverables are introduced during the kick off process, and is intended to satisfy one or more outputs for the current stage. Each draft deliverable is given a version number and placed under configuration management control. As participants review the draft deliverables, they are responsible for reporting errors found and concerns they may have to the PDR via electronic mail. The PDR in turn consolidates these reports into a series of issues associated with a specific version of a deliverable. The person in charge of developing the deliverables works to resolve these issues, and then releases another version of the deliverables for review. This process iterates until all issues are resolved for each deliverable. There is no formal check off/signature forms for this part of process. The intent here is to encourage review and feedback.

At the discretion of the PDR and PER, certain issues may be reserved for resolution in later stages of the development lifecycle. These issues are disassociated from the specific deliverable, and tagged as "open issues". Open issues are reviewed during the kickoff meeting for each subsequent stage. Once all issues against a deliverable have been resolved or moved to open status, the final (release) draft of the deliverable is prepared and submitted at the PDR. When final drafts of all required stage outputs have been received, the PDR reviews the final suite of deliverables, reviews the amount of labor expends against this stage of the project, and uses this information to update the project plan.

The project plan update includes a detailed list of tasks, their schedule and estimated level of effort for the next stage. The stage following the next stage (out stages) in the project plan are updated to include a high level estimate of schedule the level of effort, based on current project experience. Our stages are maintained at high level in

the project plan, and are included primarily for information purposes: direct experience has shown that it is very difficult to accurately plan detailed tasks and activities for our stages in a software development lifecycle. The updated project plan and schedule is a standard deliverables for each stage of the project. The PDR circulates the updated project plan and schedule for review and comment, and iterates these documents until all issues have been resolved or moved to open status. Once the project plan and schedule has been finalized, all final deliverables for the current stage are made available to all project participants, and the PDR initiates the next process.

1.4.4 In-stage Assessment Process

This is formal quality assurance review process for each stage. In a small software development project, the deliverables for each stage are generally small enough that it is not cost effective to review them for compliance with quality assurance standards before the deliverables have been fully developed. As a result, only on in-stage assessment is scheduled for each stage.

This process is initiated when the PDR schedules an in-stage assessment with the independent Quality Assurance Reviewer (QAR), a selected En-user Reviewer (usually a Subject Matter Expert), and a selected Technical Reviewer. These reviewers formally each deliverable to make judgments as to the quality and validity of the work product, as well as its compliance with the standards defined for deliverables of that class. Deliverable class standards are defined in the software quality assurance section of the project plan.

The end-user Reviewer is tasked with verifying the completeness and accuracy of the deliverable in terms of desired software functionality. The technical Reviewer determines whether the deliverable contains complete and accurate technical information.

The QA Reviewer is tasked solely with verifying the completeness and compliance of the deliverable against the associated deliverable class standard. The QAR may make recommendations, but cannot raise formal issues that do not relate to the deliverable standard.

Each reviewer follows a formal checklist during their review, indicating their level of concurrence with each review item in the checklist. Refer to the software quality assurance plan for this project for deliverable class standards and associated review checklists. A deliverable is considered to be acceptable when of the deliverable and review checklist items. Any issues raised by the reviewers against a specific deliverable will be logged and relayed to the personnel responsible for generation of the deliverable. The revised deliverable will then be released to project participants for formal review iteration. Once all issues for the deliverable have been addressed, the deliverable will be resubmitted to the reviewers for reassessment. Once all three reviewers have indicated concurrence with the deliverable, the PDR will release a final in-stage assessment report and initiate the next process.

1.4.5 Stage Exit Process

The stage exit is the vehicle for securing the concurrence of principal project participants to continue with the project and move forward into the next stage of development. The purpose of a stage exit is to allow all personnel involved with the project to review the current project plan and stage deliverables, provide a forum to raise and concerns, and to ensure an acceptable action plan exists for all open issues.

The process begins when the PDR notifies all project participants all deliverables for the current stage have been finalized and approved via the In-stage Assessment report. The PDR then schedules a stage exit review with the project executive sponsor and the PER as a minimum. All interested participants are free to attend the review as well. This meeting may be conducted in person or via Web teleconference.

The stage exit process ends with the receipt of concurrence from the designated approvers to proceed to the next stage.

CHAPTER-II

CRYPTOGRAPHY

2.1 What is Cryptography?

Cryptography is the science of writing in secret code, more generally, cryptography can be thought of as the art of mangling information into apparent unintelligibility in a manner allowing a secret method of unmangling. The basic service provided by cryptography is the ability to send information between participants in a way that prevents others from reading it. In data and telecommunications, cryptography is necessary network, particularly the Internet.

2.2 Purpose of Cryptography

Within the context of any application-to-application communication, there are some specific security requirements, including:

- **Authentication:** The process of proving one's identity. The primary forms of host-to-host authentication on the Internet today are name-based or address-based, both of which are notoriously weak.
- **Privacy/Confidentiality:** Ensuring that no one can read the message except the intender receiver.
- **Integrity:** Assuring the receiver that the received message has not been altered in any way from the original.
- **Non-repudiation:** A mechanism to prove that the sender really send this message.

A message in its original form is known as **plaintext or clear text**. The mangled information is known as **cipher text**. The process for producing cipher text from plaintext is known as **encryption** and the reverse of encryption is **decryption**.

2.3 Cryptographic Functions

There are three kinds of cryptographic function: Hash functions, secret key functions and public key function. Public key cryptography involves the use of two keys. Secret key cryptography involves the use of one key. Hash functions involve the use of zero keys.

2.3.1 Secret Key Cryptography

With secret key cryptography, a single key is used for both encryption and decryption. The sender uses the key (or some set of rules) to encrypt the plaintext and sends the cipher text to the receiver. The receiver applies the same key (or rule set) to decrypt the message and recover the plaintext. Because a single key is used for both functions, secret key cryptography is also called **symmetric encryption**.

With this form of cryptography, it is obvious that the key must be known to both the sender and the receiver; that, in fact, is the secret. The biggest difficulty with this approach, of course, is the distribution of the key.

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

Stream ciphers come in several flavors but two are worth mentioning here. *Self-synchronizing stream ciphers* calculate each bit in the key stream as a function of the previous n bits in the key stream. It is termed “self-synchronizing” because the decryption process can stay synchronized with the encryption process merely by knowing how far into the n -bit key stream it is. One problem is error propagation: a garbled bit in transmission will result in n garbled bits at the receiving side. *Synchronous stream ciphers* generate the key stream but by using the same key stream generation function at sender and receiver. While stream ciphers do not

propagate transmission errors, they are, by their nature, periodic so that the key stream will eventually repeat.

There four block ciphers commonly used are:

- Electronic Codebook(ECB) mode
- Cipher Block Chaining(CBC) mode
- Cipher Feedback(CFB) mode
- Output Feedback(OFB) mode

Block ciphers can operate in one of several modes; the following four are the most important:

- **Electronic Codebook (ECB) mode** is the simplest, most obvious application: the secret key is used to encrypt the plaintext block to form a cipher text block. Two identical plaintext blocks, then, will always generate the same cipher text block. Although this is the most common mode of block ciphers, it is susceptible to a variety of a variety of brute-force attacks.
- **Cipher Block Chaining (CBC) mode** adds a feedback mechanism to the encryption scheme. In CBC, the plaintext is exclusively-ORed (XORed) with the previous cipher text block prior to encryption. In this mode, two identical blocks of plaintext never encrypt to the same cipher text.
- **Cipher Feedback (CFB) mode** is a block cipher implementation as a self synchronizing stream cipher. CFB mode allows data to be encrypted in units smaller than the block size, which might be useful in some applications such as encrypting interactive terminal input. If we were using 1-byte CFB mode, for example, each incoming character is placed into a shift register the same size as the block, encrypted, and the block transmitted. At the receiving side, the cipher text is decrypted and the extra bits in the block(i.e., everything above and beyond the one byte) are discarded.
- **Output feedback (OFB) mode** is a block cipher implementation conceptually similar to a synchronous stream cipher. OFB prevents the same plaintext block from generating the same cipher text block by using an internal feedback mechanism that is independent of both the plaintext and cipher text bit streams.

2.3.2 Public Key Cryptography

Public-key cryptography has been said to be the most significant new development in cryptography in the last 300-400 years. Modern PKC was first described publicly by Stanford university professor Martin Hellman and graduate student Whitfield Diffie in 1976. Their paper described a 2-key cryptosystem in which two parties could engage in a secure communication over a non secure communication channel without having to share a secret key.

Generic PKC employs two keys that are mathematically related although knowledge of one key does not allow someone to easily determine the other key. One key is used to encrypt the plain text and the other key is used to decrypt the cipher text. The important point here is that it **does not matter which key is applied first**, but that both keys are required for the process to work because a pair of keys are required, this approach is also called **asymmetric cryptography**.

In PKC, one of the keys is designated the public key and may be advertised as widely as the owner wants. The other key is designated the private key and is never revealed to another party. It is straightforward to send a message under this scheme. Suppose Alice wants to send Bob a message. Alice encrypts some information using Bob's public key; Bob decrypts the cipher text using his private key. This method could be also used to prove who sent a message; Alice, for example, could encrypt some plain text with her private key; when Bob decrypts using Alice's public key, he knows that Alice sent the message and Alice cannot deny having sent the message (non repudiation).

2.3.3 Hash Functions

Hash functions, also called *message digests* and *one-way encryption*, are algorithms that, in some sense, used no key instead, a fixed-length Hash value is computed based upon plain text that makes it impossible for either the contents or length of the plain text to be recovered. Hash algorithms are typically used to provide a digital fingerprint of a file's contents often used to ensure that the file has not been altered by an intruder or virus. Hash functions are also commonly employed by many operating

systems to encrypt passwords. Hash functions, then, provide a major of the integrity of a file.

Hash functions are sometimes misunderstood and some sources claim that no two files can have the same hash value this is, in fact, not correct. Consider a hash function that provides a 128-bit hash value. There are, obviously, 2^{128} possible hash values. But there are lot more than 2^{128} possible files. Therefore, there have to be multiple files- in fact; there have to be infinite no of files. There can have the same 128-bit hash value.

We will call the hash of a message m , $h(m)$. It has the following properties:

- For any message m , it is relatively easy to compute $h(m)$. This just means that in order to be practical it can't take a lot of processing time to compute the hash.
- Given $h(m)$, there is no way to find an m that hashes to $h(m)$ in way that is substantially easier than going through all possible values of m and computing $h(m)$ for each one.
- Even though it's obvious that many different values of m will be transformed to the same value $h(m)$ (because there are many more possible values of m), it is computationally infeasible to find two values that hash to the same thing.

An example of the sort of functions that might work is taking the message m , treating it as a number, adding some large constants, squaring it and taking the middle n digits as a hash. While this would not be difficult to compute, its not obvious how you could find a message that would produces a particular hash, or how one might find two messages with the same hash.

2.4 Cryptographic Algorithms

The following important algorithms which are normally used in cryptography:

2.4.1 Secret Key Algorithm

- Data Encryption Standard (DES)

Two important variants that strengthens DES are :

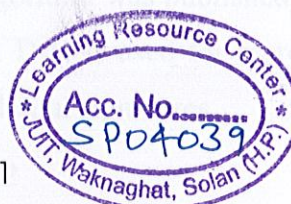
Triple –DES (3 DES): A variant of DES that employs up to three 56 bit-keys and makes three encryption/decryption passes over the block

DESX: A variant devised by Ron Rivest. By combining 64 additional key bits to the plain text prior to encryption, effectively increases the key length to 128 bits.

Other algorithm includes:

- **Advanced Encryption Standard (AES):** In cryptography the advanced encryption standard (AES), also known as Rijndael is a block cipher adopted as an encryption standard by the U.S government. It is expected to be used worldwide and analyses extensively, as was the case with its predecessors, the Data Encryption Standard (DES). AES is one of the most popular algorithms used in symmetric key cryptography. It has fixed block size of 128 bits and a key of 128,192,256 bits.
- **CAST-128/256:** CAST-128 is a DES-like substitution-permutation crypto algorithm, employing a 128-bit key operating on a 64-bit block. CAST-256 is an extension of CAST-128, using a 128-bit block size and a variable length (128,160,192,224 or 256 bit) key. CAST is named for its developers, Carlisle Adams and Stafford Tavares and is available internationally. CAST-256 was one of the Round 1 algorithms in the AES process.
- **International Data Encryption Algorithm (IDEA):** Secret-key cryptosystem written by Xuejia Lai and James Massey, in 1992 and patented by Ascom; a 64-bit SKC block cipher using a 128-bit key.
- **Blowfish:** A symmetric 64-bit block cipher invented by Bruce Schneier; optimized for 32-bit processors with large data caches, it is significantly faster than DES on a Pentium/Power PC-class machine. Key lengths can vary from 32 to 448 bits in length. Blowfish, available freely and intended as a substitute for DES or IDEA, is in use over 80 products.

- **Two fish:** A 128-bit block cipher using 128, 192 or 256 keys. Designed to be highly secure and highly flexible, well-suited for large microprocessors, 8-bit smart card microprocessors, and desiccated hardware. Designed by a team led by Bruce Schneier and was one of the Round 2 Algorithms in the AES process.
- **Camellia:** A secret-key, block-cipher crypto algorithm developed jointly by Nippon Telegraph and Telephone(NTT) Corp. and Mitsubishi Electric Corporation(MEC) in 2000. Camellia has some characteristic in common with AES: a 128-bit block size, support for 128, 192 and 256-bit key lengths, and suitability for both software and hardware implementations on common 32-bit processors as well as 8-bit processors (e.g. smart cards, cryptographic hardware, and embedded systems).
- **Mistyi:** Developed at Mitsubishi electric Crop, a block cipher using a 128-bit key and 64-bit block and a variable number of rounds. Designed for hardware and software implementations and is resistant to differential and linear cryptanalysis.
- **Secure and Fast Encryption routine (SAFER):** Secret-key crypto scheme designed for implementation in software. Versions have been defined for 40, 64 and 128 bit keys.
- **KASUMI:** A block cipher using a 128-bit key that is a part of the third generation partnership project (3gpp), formally known as the Universal Mobile communication systems.
- **SEED:** A block cipher using a 128-bit blocks and 128-bit keys. Developed by the Korea Information Security Agency(KISA) and adopted as a national standard algorithm in South Korea.



- **Skipjack:** SKC scheme proposed for Capstone. Although the details of the algorithm were never made public, Skipjack was a block cipher using an 80-bit key and 32 iteration cycles per 64-bit block.

2.4.2 Public Key Algorithms:

- **RSA:** The first, and still most common, PKC implementation, named for the three MIT mathematicians who developed it – Ronald Rivest, Adi Shamir, and Leonard Adleman. RSA today is used in hundreds of software products and can be used for key exchange, digital signatures, or encryption of small blocks of data. RSA uses a variable size encryption block and a variable size key. The key-pair is derived from a very large number, n , that is the product of two prime numbers chosen according to special rules; these primes may be 100 or more digits in length each, yielding an n with roughly twice as many digits as the prime factors. The public key information includes n and a derivative of one of the factors of n ; an attacker cannot determine the prime factors of n (and, therefore, the private key) from this information alone and that is what makes the RSA algorithm so secure. (Some descriptions of PKC erroneously state that RSA's safety is due to the difficulty in factoring large prime numbers. In fact, large prime numbers, like small prime numbers, only have two factors!) The ability for computers to factor large numbers, and therefore attack schemes such as RSA, is rapidly improving and systems today can find the prime factors of numbers with more than 200 digits. Nevertheless, if a large number is no known factorization algorithm that will solve the problem in a reasonable amount of time: a 2005 test to factor a 200-digit number took 1.5 years and over 50 years of compute time regardless, one presumed protection of RSA is that users can easily increase the key size to always stay ahead of the computer processing curve.
- **Differ-Hellman:** After the RSA algorithm was published, Diffie and Hellman came up with their own algorithm. D-H is used for secret-key key exchange only, and not for authentication or digital signatures.

- **Digital Signature Algorithm (DSA):** The algorithm specified in NIST's Digital Signature Standard (DSS), provides digital signature capability for the authentication of message.
- **ElGamal:** Designed by Taher Elgamal, a PKC system similar to Diffie Hellman and for key exchange.
- **Elliptic Curve Cryptography (ECC):** A PKC algorithm based upon elliptic curves. ECC can offer levels of security with small keys comparable to RSA and other PKC methods. It was designed for devices with limited compute power and/or memory, such as smartcards and PDAs.
- **Public-Key Cryptography Standards (PKCS):** A set of interoperable standards and guidelines for public-key cryptography, designed by RSA Data Security Inc.
 - **PKCS#1:** RSA Cryptography Standard.
 - **PKCS#2:** Incorporated into PKCS#1.
 - **PKCS#3:** Diffie-Hellman Key-Agreement Standard.
 - **PKCS#4:** Incorporated into PKCS#1.
 - **PKCS#5:** Password-Based Cryptography Standard.
 - **PKCS#6:** Extended-Certificate Syntax Standard.
 - **PKCS#7:** Cryptographic Message Syntax Standard.
 - **PKCS#8:** Private-Key Information Syntax Standard
 - **PKCS#9:** Selected Attribute Types.
 - **PKCS#10:** Certification Request Syntax Standard.
 - **PKCS#11:** Cryptographic Token Interface Standard
 - **PKCS#12:** Personal Information Exchange Syntax Standard.
 - **PKCS#13:** Elliptic Curve Cryptographic Standard.
 - **PKCS#14:** Pseudorandom Number Generation Standard is no longer available.

- **PKCS#15:** Cryptographic Token Information Format Standard.
- **Cramer-Shoup:** A public-key cryptosystem proposed by R.Cramer and V. Shoup of IBM in 1998.
- **Key Exchange Algorithm (KEA);** A variation on Diffie-Hellman; proposed as the key exchange method for Capstone.
- **LUC:** A public-key cryptosystem designed by P.J. Smith and based on Lucas sequences. Can be used for encryption and signatures, using integer factoring.

2.4.3 Hash Functions:

Hash algorithms that are in common use today include:

- **Message Digest (MD) Algorithms:** A series of byte-oriented algorithms that produce a 128-bit hash value from an arbitrary-length message.
- **MD2:** Designed for system with limited memory, such as smart cards.
- **MD4:** Developed by Rivest, similar to MD2 but designed specifically for fast processing in software.
- **MD5:** Also developed by Rivest after potential weaknesses were reported in MD4; this scheme is similar to MD4 but is slower because more manipulation is made to the original data. MD5 has been implemented in a large number of products although several weaknesses in the algorithm were demonstrated by German cryptographer Hans Dobbertin in 1996.
- **Secure Hash Algorithm (SHA):** Algorithm for NIST's Secure Hash Standard (SHS). SHA-1 produces a 160-bit hash value and it describes five algorithms in the SHS: SHA-1 plus SHA-224, SHA-256, SHA-384, and SHA-512 which can produce hash values that are 224, 256, 384, or 512 bits in length respectively.

- **RIPEMD:** A series of message digests that initially came from the RIPE (Race Integrity Primitive Evaluation) project. RIPEMD-160 was designed by Hans Dobbertin, Antoon Bosselaers, and Bart Preneel, and optimized for 32-bit processors to replace the current 128-bit hash functions. Other versions include RIPEMD-256, RIPEMD-320, and RIPEMD-128.
- **HAVEL:** Designed by Y. Zheng, J. Pieprzyk and J. Seberry, a hash algorithm with many levels of security. HAVEL can create hash values that are 128, 160, 192, 224, or 256 bits in length.
- **Whirlpool:** A relatively new hash function, designed by V. Rijimen and P.S.L.M. Barreto. Whirlpool operates on message less than 2^{256} bits in length, and produces a message digest of 512 bits. The design of this has function is very different than that of MD5 and SHA-1, making it immune to the same attacks as on those hashes.
- **Tiger:** Designed by Ross Anderson and Eli Biham, Tiger is designed to be secure, run efficiently on 64-bit processors, and easily replace MD4, MD5, SHA and SHA-1 in other applications. Tiger/192 produces a 192-bit output and is compatible with 64-bits, respectively, to provide compatibility with the other hash functions mentioned above.

Certain extensions of hash functions are used for a variety of information security and digital forensics applications, such as:

- **Hash Libraries** are sets of hash values corresponding to known files. A hash library of known good files, for example, might be a set of files known to be a part of an operating system, while a hash library of known bad files might be of a set of known child pornographic images.
- **Rolling Hashes** refer to a set of hash values that are computed based upon a fixed-length “sliding window” through the input. As an example, a hash

value might be computed on bytes 1-10 of a file, then on bytes 2-11, 3-12, 4-13, etc.

- **Fuzzy Hashes** are an area of intense research and represent hash values that represent two inputs that are similar. Fuzzy hashes are to detect document, images or other files that are close to each other with respect to the content.

2.5 Advance Encryption Standard:

The algorithm used in encryption & decryption process is Rijndael, more specifically the Advance Encryption Standard. It is a secret key algorithm which takes three different key lengths for encryption & decryption, 128, 192 & 256 bits.

2.5.1 Algorithm Specification

For the AES algorithm, **the length of the input block, the output block and the State is 128 Bits**. This is represented by $Nb = 4$, which reflects the number of 32-bit words (number of columns) in the State.

For the AES algorithm, **the length of the Cipher Key, K , is 128, 192, or 256 bits**. The key length is represented by $Nk = 4, 6, \text{ or } 8$, which reflects the number of 32-bit words (number of columns) in the Cipher Key.

For the AES algorithm, the number of rounds to be performed during the execution of the algorithm is dependent on the key size. The number of rounds is represented by Nr , where $Nr =$

10 when $Nk = 4$, $Nr = 12$ when $Nk = 6$, and $Nr = 14$ when $Nk = 8$.

For both its Cipher and Inverse Cipher, the AES algorithm uses a round function that is composed of four different byte-oriented transformations: 1) byte substitution using a substitution table (S-box), 2) shifting rows of the State array by different offsets, 3) mixing the data within each column of the State array, and 4) adding a Round Key to the State.

	Key Length (<i>Nk</i> words)	Block Size (<i>Nb</i> words)	Number of Rounds (<i>Nr</i>)
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

2.5.2 Cipher

At the start of the Cipher, the input is copied to the State array. After an initial Round Key addition, the State array is transformed by implementing a round function 10, 12, or 14 times (depending on the key length), with the final round differing slightly from the first $Nr - 1$ rounds. The final State is then copied to the output. The round function is parameterized using a key schedule that consists of a one-dimensional array of four-byte words derived using the Key Expansion routine.

The individual transformations:- **SubBytes()**, **ShiftRows()**, **MixColumns()**, and **AddRoundKey()** – process the State and are described in the following subsections.

In the pseudo code the array **w[]** contains the key schedule and all Nr rounds are identical with the exception of the final round, which does not include the **MixColumns()** transformation.

The Cipher is described in the following pseudo code:

Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])

begin

byte state[4,Nb]

state = in

AddRoundKey(state, w[0, Nb-1])

for round = 1 step 1 to Nr-1

SubBytes(state)

ShiftRows(state)

MixColumns(state)

AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])

end for

SubBytes(state)

ShiftRows(state)

AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

out = state

end

2.5.3 SubBytes() Transformation

The **SubBytes()** transformation is a non-linear byte substitution that operates independently on each byte of the State using a substitution table (S-box).

This S-box, which is invertible, is constructed by composing two transformations:

1. Take the multiplicative inverse in the finite field $GF(2^8)$, the element $\{00\}$ is mapped to itself.
2. Apply the following affine transformation (over $GF(2)$):

$$b'_i = b_i \oplus b_{(i+4) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+6) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus c_i$$

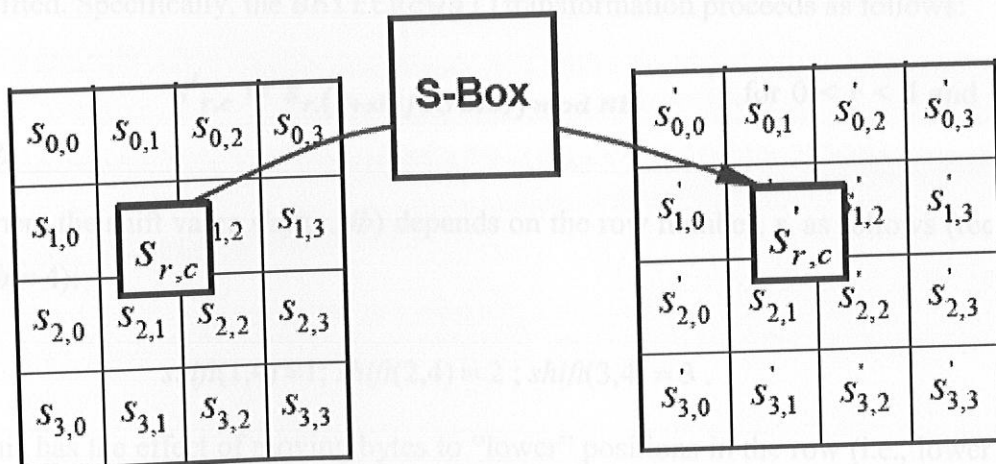
for $0 \leq i < 8$, where bi is the i^{th} bit of the byte, and ci is the i^{th} bit of a byte c with the value $\{63\}$ or $\{01100011\}$. Here and elsewhere, a prime on a variable (e.g., b') indicates that the variable is to be updated with the value on the right.

In matrix form, the affine transformation element of the S-box can be expressed as

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

S-box: substitution values for the byte xy (in hexadecimal format)

The following figure illustrates the effect of the **SubBytes()** transformation on the State:



SubBytes() applies the S-box to each byte of the State

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

S-box: substitution values for the byte xy (in hexadecimal format).

2.5.4 ShiftRows() Transformation

In the `ShiftRows()` transformation, the bytes in the last three rows of the State are cyclically shifted over different numbers of bytes (offsets). The first row, $r = 0$, is not shifted. Specifically, the `ShiftRows()` transformation proceeds as follows:

$$s'_{r,c} = s_{r,(c+shift(r,Nb)) \bmod Nb} \quad \text{for } 0 < r < 4 \text{ and } 0 \leq c < Nb$$

Nb

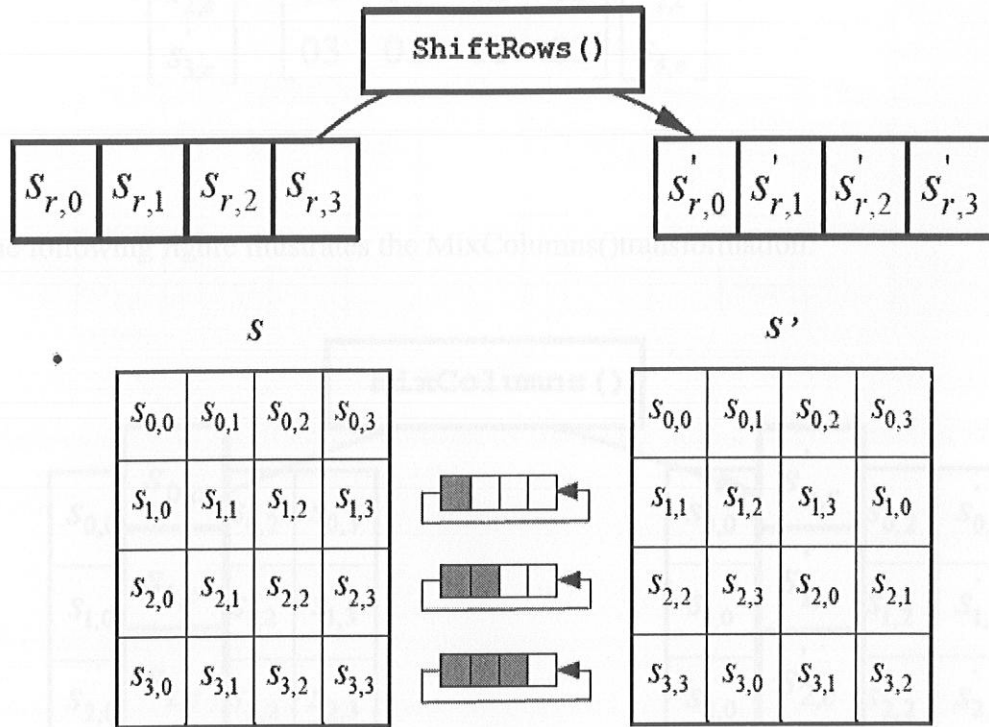
where the shift value $shift(r,Nb)$ depends on the row number, r , as follows (recall that $Nb = 4$):

$$shift(1,4) = 1; shift(2,4) = 2; shift(3,4) = 3.$$

This has the effect of moving bytes to “lower” positions in the row (i.e., lower values of c in a given row), while the “lowest” bytes wrap around into the “top” of the row (i.e., higher values of

c in a given row).

The following figure illustrates the `ShiftRows()` transformation:



`ShiftRows()` cyclically shifts the last three rows in the State

2.5.5 MixColumns() Transformation

The `MixColumns()` transformation operates on the State column-by-column, treating each column as a four-term polynomial. The columns are considered as polynomials over $GF(2^8)$ and multiplied modulo $x^4 + 1$ with a fixed polynomial $a(x)$, given by:

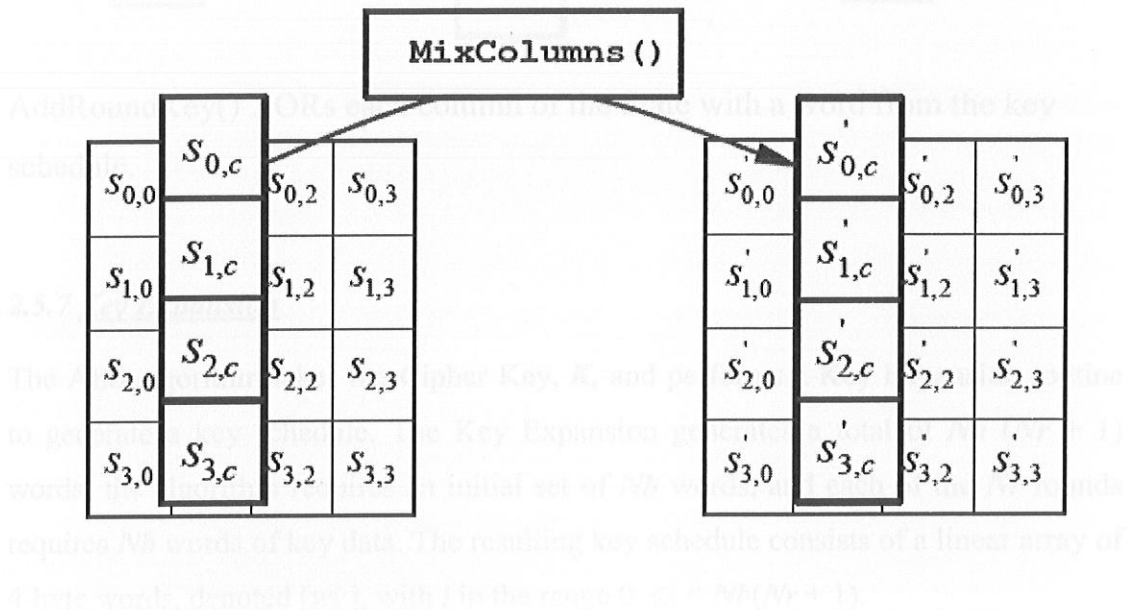
$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}.$$

This can be written as a matrix multiplication. Let

$$s'(x) = a(x) \otimes s(x) :$$

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \quad \text{for } 0 \leq c < Nb.$$

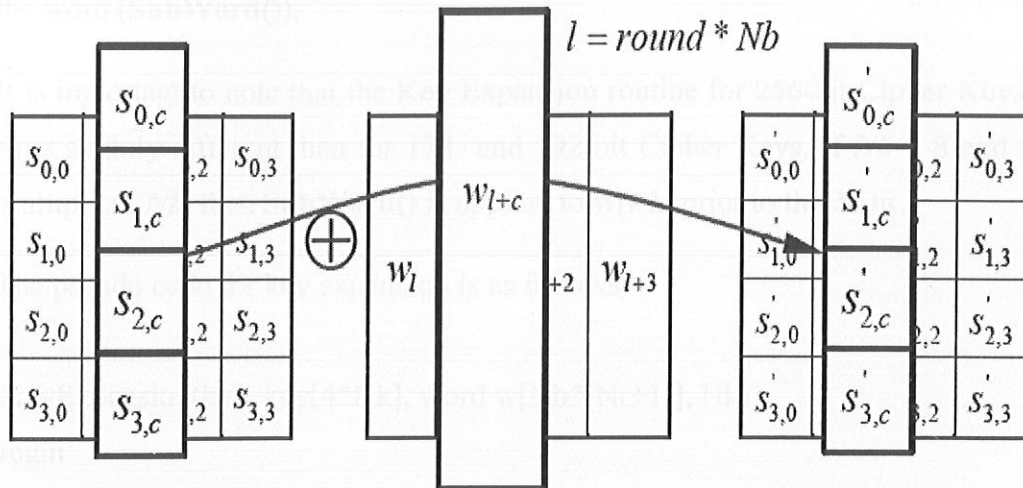
The following figure illustrates the MixColumns() transformation:



2.5.6 AddRoundKey() Transformation

In the **AddRoundKey()** transformation, a Round Key is added to the State by a simple bitwise XOR operation. Each Round Key consists of Nb words from the key schedule.

The action of this transformation is illustrated in following figure, where $l = round * Nb$.



AddRoundKey() XORs each column of the State with a word from the key schedule.

2.5.7 Key Expansion

The AES algorithm takes the Cipher Key, K , and performs a Key Expansion routine to generate a key schedule. The Key Expansion generates a total of Nb ($Nr + 1$) words: the algorithm requires an initial set of Nb words, and each of the Nr rounds requires Nb words of key data. The resulting key schedule consists of a linear array of 4-byte words, denoted $[w_i]$, with i in the range $0 \leq i < Nb(Nr + 1)$.

SubWord() is a function that takes a four-byte input word and applies the S-box to each of the four bytes to produce an output word. The function **RotWord()** takes a word $[a_0, a_1, a_2, a_3]$ as input, performs a cyclic permutation, and returns the word $[a_1, a_2, a_3, a_0]$. The round constant word array, **Rcon**[i], contains the values given by $[x^{i-1}, \{00\}, \{00\}, \{00\}]$, with x^{i-1} being powers of x (x is denoted as $\{02\}$) in the field $GF(2^8)$, (note that i starts at 1, not 0). It can be seen from the pseudo code that the first Nk words of the expanded key are filled with the Cipher Key. Every following word, $w[i]$, is equal to the XOR of the previous word, $w[i-1]$, and the word Nk positions earlier, $w[i-Nk]$. For words in positions that are a multiple of Nk , a transformation is applied to $w[i-1]$ prior to the XOR, followed by an XOR with a round constant, **Rcon**[i]. This transformation consists of a cyclic shift of the bytes in a

word (**RotWord()**), followed by the application of a table lookup to all four bytes of the word (**SubWord()**).

It is important to note that the Key Expansion routine for 256-bit Cipher Keys ($Nk = 8$) is slightly different than for 128- and 192-bit Cipher Keys. If $Nk = 8$ and $i-4$ is a multiple of Nk , then **SubWord()** is applied to $w[i-1]$ prior to the XOR.

The pseudo code for key expansion is as follows:

```

KeyExpansion(byte key[4*Nk], word w[Nb*(Nr+1)], Nk)
begin
    word temp
    i = 0
    while (i < Nk)
        w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
        i = i+1
    end while
    i = Nk
    while (i < Nb * (Nr+1))
        temp = w[i-1]
        if (i mod Nk = 0)
            temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
        else if (Nk > 6 and i mod Nk = 4)
            temp = SubWord(temp)
        end if
        w[i] = w[i-Nk] xor temp
        i = i + 1
    end while
end

```

2.5.8 Inverse Cipher

The Cipher transformations can be inverted and then implemented in reverse order to produce a straightforward Inverse Cipher for the AES algorithm. The individual transformations used in the Inverse Cipher - **InvShiftRows()**, **InvSubBytes()**, **InvMixColumns()**, and **AddRoundKey()** – process the State and are described in the following subsections. The Inverse Cipher is described in the pseudo code, where the array **w[]** contains the key schedule.

InvCipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])

begin

byte state[4,Nb]

state = in

AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

for round = Nr-1 step -1 downto 1

 InvShiftRows(state)

 InvSubBytes(state)

 AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])

 InvMixColumns(state)

end for

InvShiftRows(state)

InvSubBytes(state)

AddRoundKey(state, w[0, Nb-1])

out = state

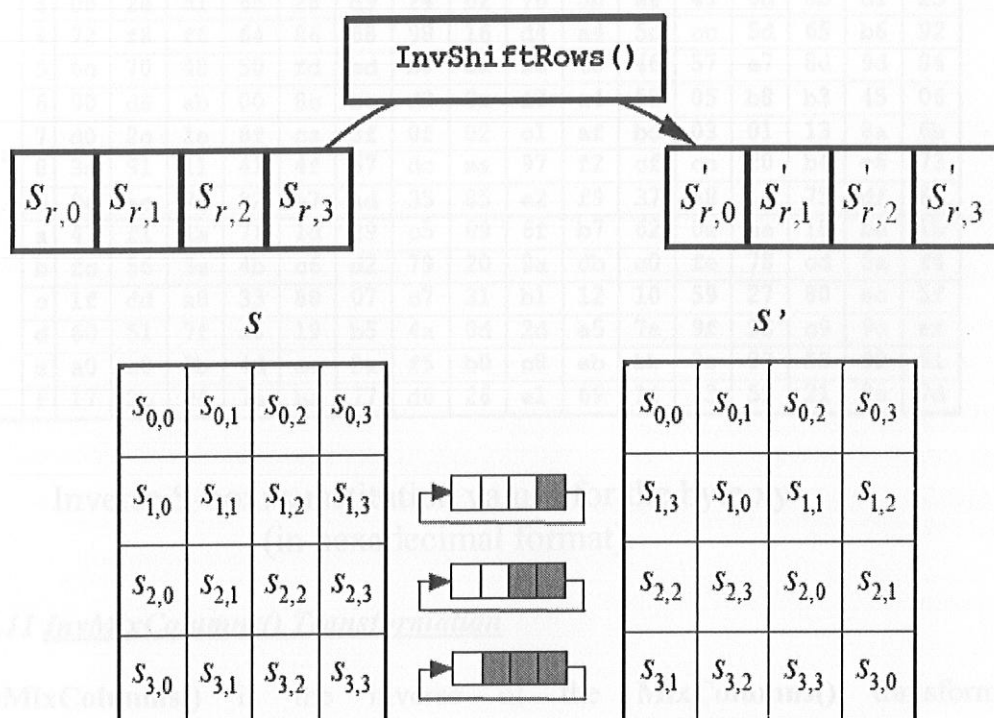
end

2.5.9 InvShiftRows() Transformation

InvShiftRows() is the inverse of the **ShiftRows()** transformation. The bytes in the last three rows of the State are cyclically shifted over different numbers of bytes (offsets). The first row, $r = 0$, is not shifted. The bottom three rows are cyclically shifted by Nb - $shift(r, Nb)$

bytes, where the shift value $shift(r, Nb)$ depends on the row number.

The following figure illustrates the **InvShiftRows()** transformation:



InvShiftRows() cyclically shifts the last three rows in the State

2.5.10 InvSubBytes() Transformation

InvSubBytes() is the inverse of the byte substitution transformation, in which the inverse S-box is applied to each byte of the State. This is obtained by applying the inverse of the affine transformation followed by taking the multiplicative inverse in $GF(2^8)$.

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
	1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
	2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
	3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
	4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
	5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
	6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
	7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
	8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
	9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
	a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
	b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
	c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
	d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
	e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
	f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

Inverse S-box: substitution values for the byte xy
(in hexadecimal format).

2.5.11 InvMixColumns() Transformation

InvMixColumns() is the inverse of the **MixColumns()** transformation. **InvMixColumns()** operates on the State column-by-column, treating each column as a four term polynomial. The columns are considered as polynomials over $GF(2^8)$ and multiplied modulo $x^4 + 1$ with a fixed polynomial $a^{-1}(x)$, given by

$$a^{-1}(x) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\}.$$

This can be written as a matrix multiplication. Let

$$s'(x) = a^{-1}(x) \otimes s(x) :$$

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \quad \text{for } 0 \leq c < Nb.$$

2.5.12 Inverse of the AddRoundKey() Transformation

AddRoundKey(), is its own inverse, since it only involves an application of the XOR operation.

2.5.13 Advantages

Implementation aspects:

- Rijndael can be implemented to run at speeds unusually fast for a block cipher on a Pentium (Pro). There is a trade-off between table size/performance.
- Rijndael can be implemented on a Smart Card in a small amount of code,
- Using a small amount of RAM and taking a small number of cycles. There is some ROM/performance trade-off.
- The round transformation is parallel by design, an important advantage in future processors and dedicated hardware.
- As the cipher does not make use of arithmetic operations, it has no bias towards big or little endian processor architectures.

Simplicity of Design:

- The cipher is fully “self-supporting”. It does not use of another cryptographic component, S-boxes “lent” from well-reputed ciphers, bits obtained from Rand tables, digits of π or other such jokes.
- The cipher does not base its security or part of it on obscure and not well understood interactions between arithmetic operations.
- The tight cipher design does not leave enough room to hide a trapdoor.

Variable block length:

- The block lengths of 192 and 256 bits allow the construction of a collision resistant iterant hash function using Rijndael as compression function. The block length of 128 bits is not considered sufficient for this purpose nowadays.

Extensions:

- The design allows the specification of variants with the block length and key length both ranging from 128 to 256 bits in steps of 32 bits.
- Although the number of rounds of Rijndael is fixed in the specification, it can be modified as a parameter in case of security problems.

2.5.14 Disadvantages

The limitations of the cipher have to do with its inverse:

- The inverse cipher is less suited to be implemented on a smart card than the cipher itself: it takes more code and cycle. (Still, compared with other ciphers, even the inverse is very fast).
- In software, the cipher and its inverse make use of different code and/or tables.
- In hardware, the inverse cipher can only partially re-use the circuitry that implements the cipher.

CHAPTER-III

PROJECT PLANNING

Effective management of a software project depends on the thoroughly planning the progress of the project. The project manager must anticipate problems which might arise and prepare tentative solutions to those problems. A plan, drawn up at the start of a project, should be used as the driver for the project. The initial plan should be the best possible plan given the available information. It evolves as the project progresses and better information becomes possible.

A structure for a software development plan is described below. As well as a project plan, managers may also have to draw up other types of plan.

Plan	Description
Quality Plan	Describes the quality procedures and standards that will be used in a project.
Validation Plan	Describes the approach, resources and schedule use for system validation.
Configuration Management Plan	Describes the configuration management procedures and structures to be used.
Maintenance plan	Predicts the maintenance requirements of the system, maintenance costs and effort required
Staff Development Plan	Describes how the skills and experience of the project team members will be developed

3.1 Project Plan

The project plan sets out the resources available to the project, the work break down and a schedule for carrying out the work. In some organizations, the project plan is a single document including all the different types of plan introduced above. In other cases, the project plan is solely concerned with the development process. Below described is one such plan we followed:

1. *Introduction:* this briefly describes the objectives of the project and sets out the constraints (e.g. budget, time etc.) which affect the project management.
2. *Project Organization:* this describes the way in which the development team is organized, the people involved and their roles in the team.
3. *Hardware and Software Requirement:* this describes the hardware and the support software required to carry out development. if hardware has to be brought, estimates of the prices and the delivery schedule should be included.
4. *Work Breakdown:* This describes the breakdown of the project into activities and identifies the milestones and deliverables associated with each activity.
5. *Project Schedule:* This describes the dependencies between activities, the estimated time required to each milestone and the allocation of people to activities.
6. *Monitoring and Reporting Mechanism:* This describes the management reports which should be produced, when these should be produced and the project monitoring mechanisms used.

The project plan should be regularly revised during the project. Some parts, such as the project schedule, will change frequently; other parts will be more stable. A document organization which allows for straightforward replacement of sections should be used.

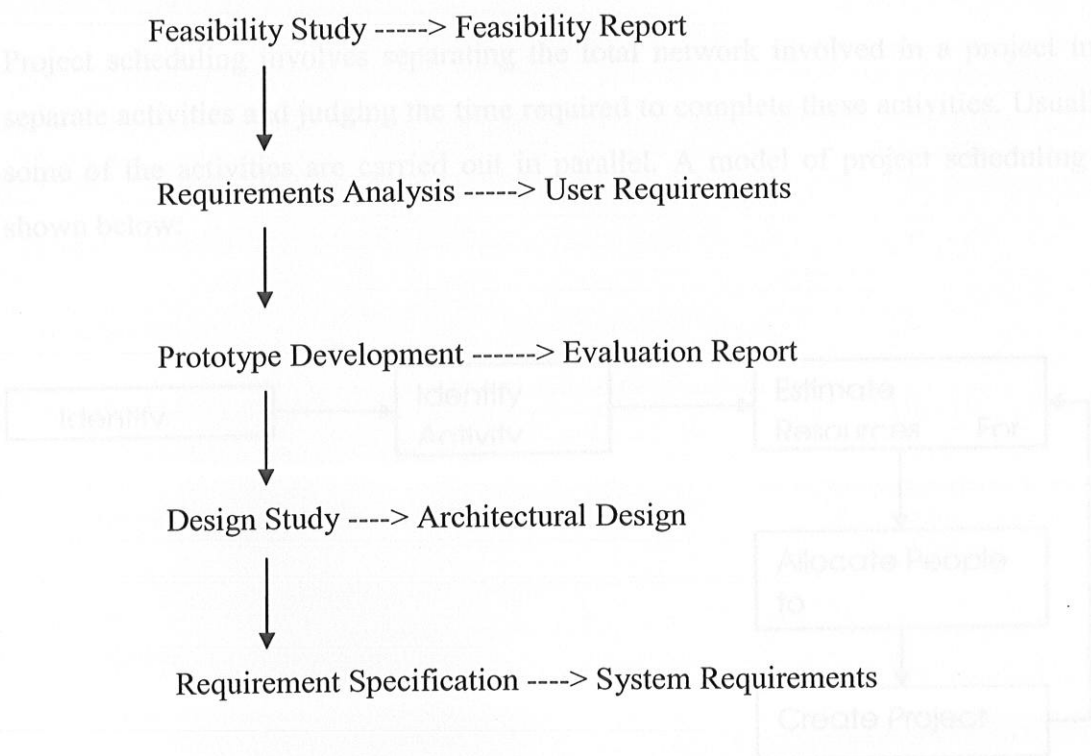
3.2 Milestones and Deliverables

When planning a project, a series of milestones should be established where a milestone is an end point of a software process activity. At each milestone, there should be a formal output, such as a report, that can be presented to management. Milestones reports need not be large documents. They may simply be a short report of

achievements in a project documents. Milestones should represent the end of a distinct, logical stage in the project.

A *deliverable* is a project result that is delivered to the customer. It is usually delivered at the end of some major project phase such as specifications, design, etc. deliverables are usually milestones but milestones may be internal project result that are used by the project manager to check project progress but which are not delivered to the customer.

Following is the model of the activities normally processed in requirements specifications:

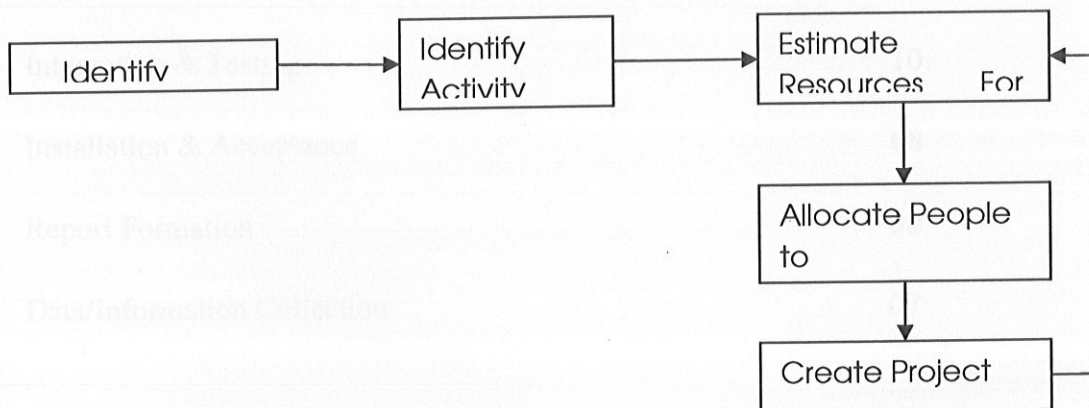


3.3 Project Scheduling

Project scheduling is a particularly demanding task for software managers. Manager's estimate and resource required to complete activities and organize them in a coherent sequence. Unless the project being scheduled is similar to a previous project, previous estimates are an uncertain basis for new project scheduling. Schedule estimation is further complicated by the fact that different projects may use different design methods and implementation languages.

If a project is technically advanced, initial estimates will almost certainly be optimistic even when the managers try to consider all eventualities. In this respect, software scheduling is no different from scheduling any other type of large advanced project.

Project scheduling involves separating the total network involved in a project into separate activities and judging the time required to complete these activities. Usually, some of the activities are carried out in parallel. A model of project scheduling is shown below:



3.4 Bar Charts & Activity Networks

Bar charts and activity networks are graphical notations which are used to illustrate the project schedule. Bar charts show who is responsible for each activity and when the activity is scheduled to begin and end. Activity networks show the dependencies between the different activities making up a project.

Following is the activity network we passed by in designing and development of the software product.

Task	Duration (Days)
Initial Discussion	03
Project Planning	11
Requirements Analysis	03
Design	20
Implementation	25
Integration & Testing	10
Installation & Acceptance	08
Report Formation	30
Data/Information Collection	07

CHAPTER-IV

REQUIREMENT ANALYSIS

The problems that software engineers have to solve are often immensely complex. Understanding the nature of the problems can be very difficult, especially if the system is new. Consequently, it is difficult to establish exactly what the system should do. The descriptions of the services and constraints are the *requirements* for the system and the process of finding out, analyzing documenting and checking the services and constraints is called *requirements engineering*.

The requirement engineering process is composed of the following 3 phases:

- *User requirement analysis*: User requirements are statements, in a natural language plus diagrams, of what services is expected to provide and the constraints under which it must operate.
- *System Requirement Analysis*: System Requirements set out the system services and constraints in detail. The system requirements document, which is sometimes called functional specifications, should be precise. It may serve as a contract between the system buyer and software developer.
- *Requirements Elicitation Analysis*: Requirements elicitation phase involves the technical software development team to find out about the application domain, what services the system should provide, the required performance of the system, hardware constraints and so on.

4.1 User Requirements

User requirements are the statements which provide the services the system is expected to provide. Following are some of the user requirements:

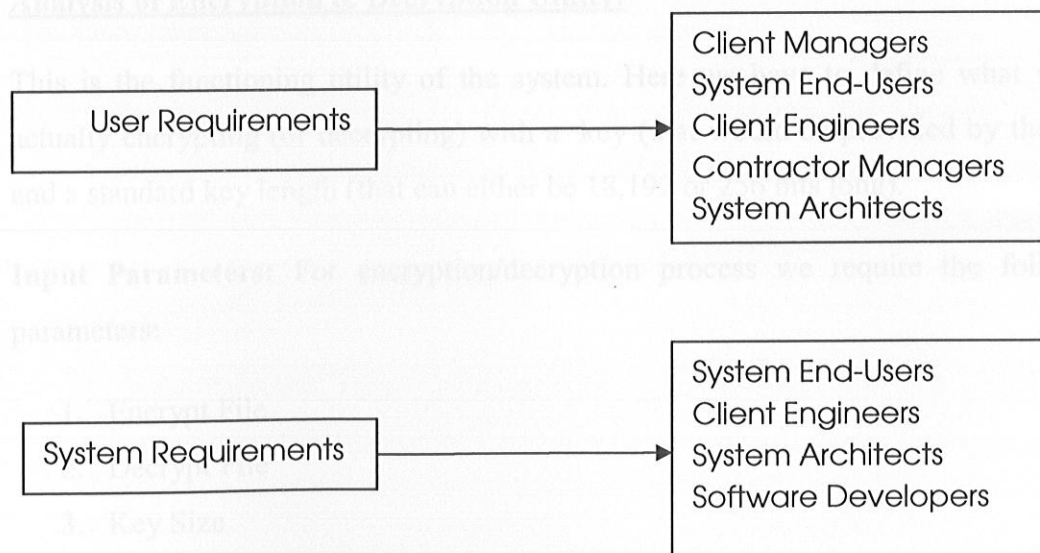
- The software must provide the means of encrypting and decrypting files selected by the user.

4.2 System Requirements

1. The user should be provided with the facilities to provide files of the following types:
 - Text files
 - Document Files
 - Audio Files
 - Picture files
 - Video Files
2. The user should be able to encrypt/decrypt files selected by him/her.
3. The user should be asked to provide the password required for encryption of files and the same password should be used decrypt that encrypted file.
4. The user should be provided with three different encryption and decryption key length, namely 128 bits, 192 bits and 256 bits.

4.3 Different Readers

Following are the readers of different types of specifications mentioned above:



4.4 Functional and Non-functional Requirements

Software system requirements are often classified as functional and non-functional requirements:

- *Functional Requirements*: These are statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations. In some cases, the functional requirements may also explicitly state what the system should not do.
- *Non functional Requirements*: these are constraints on the services or functions offered by the system. They include timing constraints, constraints on the development process, standards, etc.

The *functional requirements* for a system describe the functionality or services that system is expected to provide. These depend on the type of the software and the type of the system which is being developed. Functional system requirements describe the system function in detail, its input and output, exceptions, etc.

Upon analysis of the problem definition, our team decided to propose the functional requirement of a utility for encryption and decryption of files using standard algorithm with standard key length.

Analysis of Encryption & Decryption Utility:

This is the functioning utility of the system. Here we have to define what we are actually encrypting (or decrypting) with a key (that would be provided by the user) and a standard key length (that can either be 18,192 or 256 bits long).

Input Parameters: For encryption/decryption process we require the following parameters:

1. Encrypt File
2. Decrypt File
3. Key Size

The first two parameters are used to encrypt and decrypt files. The third parameter, key size is chosen as desired by our need.

Output Delivered: When encrypting and decrypting files, the user provide the system with a password while encrypting a file, and as output he/she gets the encrypted file, and while decrypting the file, he/she provides the same password which was given in encryption process and as output the decrypted (original) file is achieved.

Function: The user can encrypt & decrypt any text, document, audio, video, picture file with any of the three key lengths provided by the system.

Non-functional requirements, as the name suggests, are those requirement which are not directly concerned with the specific functions delivered by the system. They may relate to emergent system properties such as, reliability, response time and store occupancy. Alternatively they may define constraints on the system such as capabilities of I/O devices and the data representations used in system interfaces.

Many non-functional requirements relate to the system as a whole rather than to individual system failures. This means that they are often more critical than individual functional requirement. While failure to meet an individual function requirement may degrade the system, failure to meet an individual functional requirement may take the whole system unusable. For example, if an aircraft system does not meet its reliability requirements, it will not be certified as safe for operation; if a real time control system fails to meet its performance requirements, the control functions will not operate correctly.

The there main types of functional requirements are:

- 1) **Product Requirements:** These are requirements that specify product behavior. Examples include *performance requirements* on how fast the system must execute ad much memory it requires, *reliability requirements* that set out the acceptable failure rate, *portability requirements* and *usability requirements*.
- 2) **Organizational Requirements:** These are derived from the policies and procedures in the customer's and developer's organization. Example includes process standards which must be used; implementation requirements such as the programming language or design and method used; and delivery

requirements which specify when the product and its documentation are to be delivered.

- 3) **External Requirements:** this broad heading covers all requirements which are derived from factors external to the system and its development process. These include *legislative requirements* which must be followed to ensure that the system operates within the law and *ethical requirements* which are placed on a system to ensure that it will be acceptable to its users and general public.

Product Requirements:

Performance Requirement:

- Minimum CPU Speed: 166 Mhz
- Minimum HDD: 96 MB
- Minimum RAM: 32 MB

Organizational Requirement:

- Encryption/Decryption Algorithm Used: Advance Encryption Standard
- Programming Language: Java 1.6.1_10
- Operating System: Windows XP
- Platform Used: Windows.

4.5 Requirement Elicitation Analysis

After initial feasibility studies, the next stage of the requirements engineering process is requirements elicitation analysis. Requirements elicitation analysis may involve a variety of different kinds of people in an organization. The term *stakeholder* is used to refer anyone who should have some direct or indirect influence on the system requirement. Stakeholders include end-users who will interact with the system and everyone else in an organization who will be affected by it. A generic process model of the elicitation analysis is shown below:

- 1) *Domain understanding*: Analyst must develop their understanding of the application domain.
- 2) *Requirement Collection*: this is the process of interacting with stakeholders in the system to discover their requirements. Obviously, domain understanding develops further during this activity.
- 3) *Classification*: This activity takes the unstructured collection of requirements and organizes them into coherent structures.
- 4) *Conflict resolution*: Inevitably, where multiple stakeholders are involved, requirements will conflict. This activity is concerned with finding and resolving these conflicts.
- 5) *Prioritization*: In any set requirements some will be more important than others. This stage involves interaction with stakeholders to discover the most important requirements.
- 6) *Requirements Checking*: The requirements are checked to discover if they are complete, consistent and in accordance with what stakeholders really want from the system.

4.6 Problem Definition

Statement of Purpose:

Statement of purpose (SOP) is a concise, textual statement of what the system does, it may extend to one paragraph. The statement of purpose for our system is defined below.

To propose a system which can encrypt and decrypt selected files using standard encryption and decryption algorithm to protect systems (or users) against unauthorized users.

CHAPTER-V

DESIGN

Large systems are always decomposed into the sub-systems that provide some related set of services. The initial design process of identifying these sub-systems and establishing a framework for sub-systems control and communication is called *design process* and the output of this design process is a description of the *software architecture*.

In our analysis of software requirements our team decided to design them in the following two phases:

- 1) **Environmental Model:** Environmental Model gives the system environment boundary. It is composed of the following components:
 - a) **Statement of Purpose:** Concise, textual statement of what the system does, it may extend to one paragraph.
 - b) **Context Diagram:** It sets the context of the system in the environment. It is decomposed of *terminators* with the system communicates, *data stores* shared by the *system* and *terminators* and data received from and sent to the system.
 - c) **Event List:** A list of stimuli coming from the environment to which the system must respond. It is composed of *normal events*, *temporal events* and *control events*
- 2) **Behavioral Model:** Behavioral Model tells the behavior displayed by the internal of the system. It is composed of the following components:
 - a) **Cartesian Hierarchy:** Define a function for each even list having the same response.
 - b) **Data Flow Diagrams:** The aim of data flow diagrams is to tell what functions the system must perform with their interactions.
 - c) **Process Specifications:** A description of what is happening a bottom-level primitive function.

- d) Data dictionary: Contains a listing and description of all data items and meaning of flow/stores.
- e) User-Interface Design: This lets the software programmers decide the user interface design of the software product in accordance with client requirements.

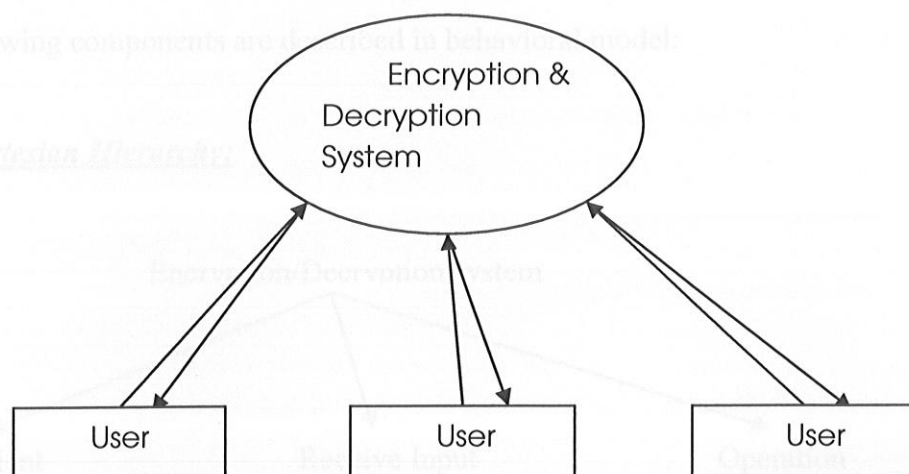
5.1 Environmental Model

The following components are described in the environmental model:

5.1.1 Statement of Purpose:

Statement of Purpose (SOP) is a concise, textual statement of what the system does. It may extend to one paragraph. The system of purpose for our system is defined below

5.1.2 Context Diagram:



The terminator 'User' refers to encryption & decryption of files, with the incoming arrow in the system referring to input files provided by the user and the outgoing arrow from the system referring to the encrypted files.

5.1.3 Event List:

Normal Events:

- a) User encrypts a file.
- b) User decrypts a file.

Remind Events:

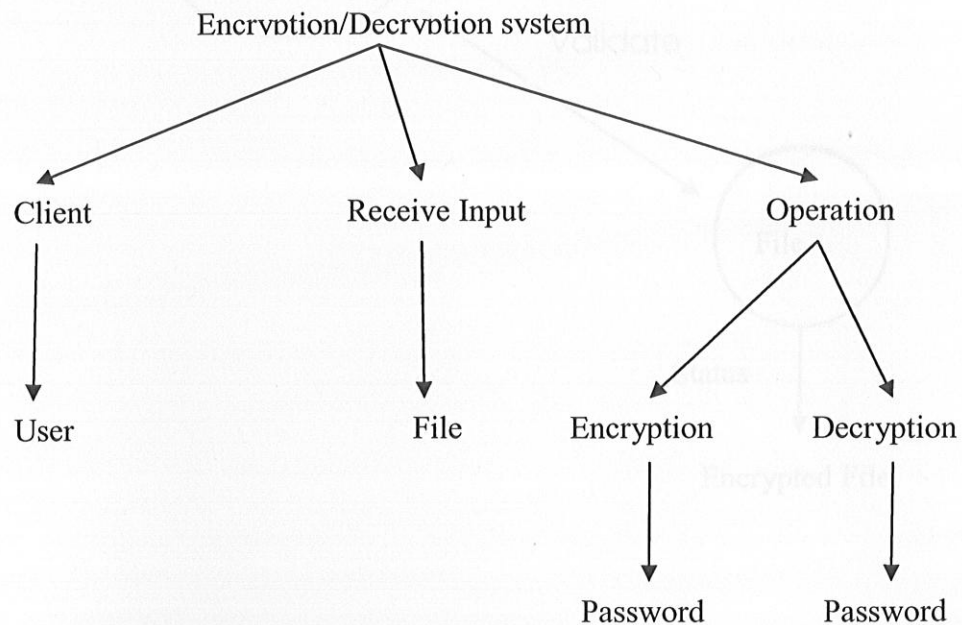
- a) Remind user to provide the correct key length.
- b) Remind the user to provide the correct password.
- c) Remind the user to decrypt an encrypted file only.

The above remind events are also applicable for administrator using the system.

5.2 Behavioral Model

The following components are described in behavioral model:

5.2.1 Cartesian Hierarchy:

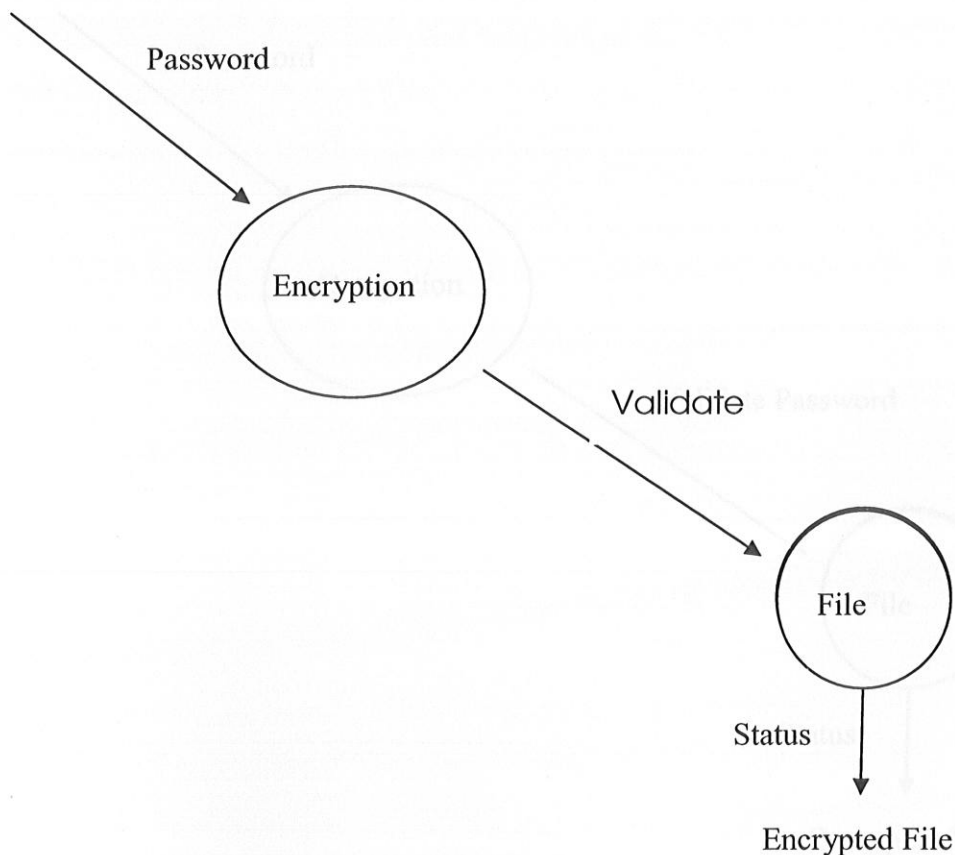


The above Cartesian Hierarchy shows the root node being Encryption & Decryption System, the level contains Client who can be a normal user receive input which can any file and the operation which can be either encryption or decryption.

5.2.2 Data Flow Diagrams:

Data flow diagrams are made from the Cartesian Hierarchy itself where we design main modules of the software system. Upon analysis, we found it necessary to provide two data flow diagrams, one for encryption of files and second for decryption of files.

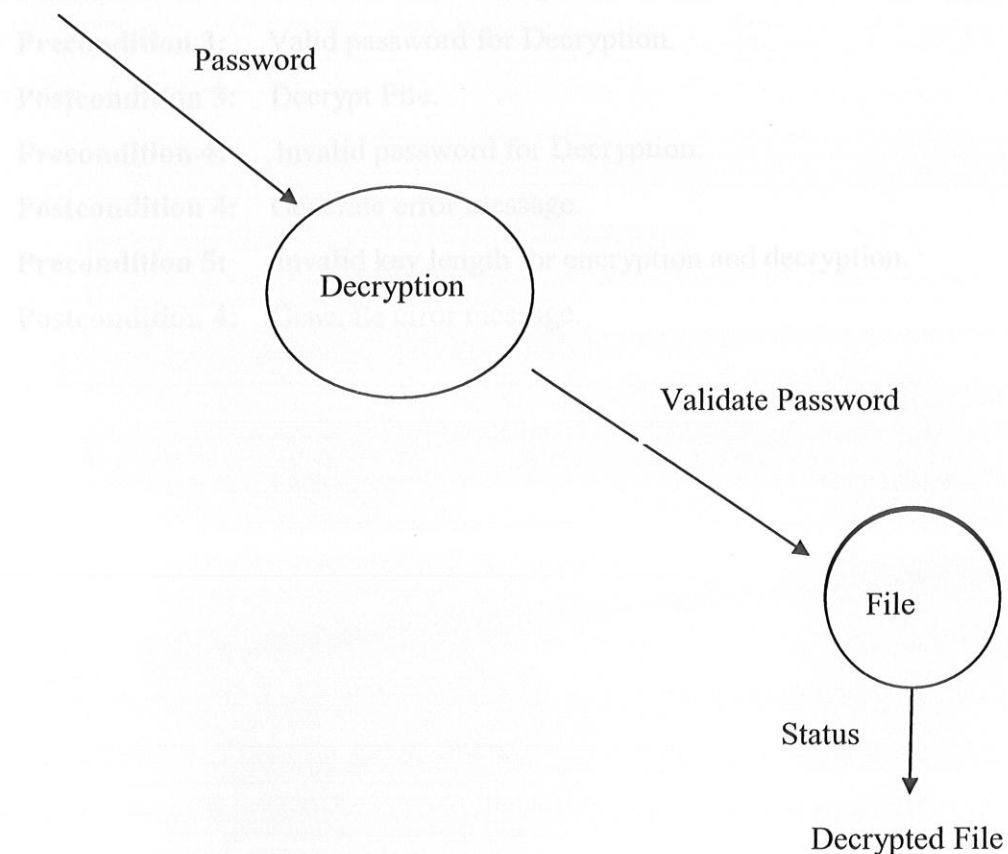
Data flow Diagram for File Encryption:



The above data flow diagram shows the main system, i.e. the encryption system which takes a password for encryption process. It then validates the password and the user selects the file for encryption. After this software encrypts the file in the end we get the encrypted file.

Data Flow Diagram for File Decryption:

For file decryption, the data flow diagram is almost same as shown above with the exception that instead of file encryption, the system would now decrypt the file. It would also take a password that is the password which should be the same password as in the encryption process. Upon successful validation of password it would decrypt the file provided by the user. The data flow diagram for this process is shown below:



5.2.3 Process Specification:

Process specification tells us that what is happening inside a bottom-level primitive function. There is no commitment to the choice of algorithm, so we can choose any way to describe process specification from given standard ones. The algorithm that our team chooses is *pre/post Condition*. Here we write conditions in which certain important processes are carried out and compliment them by writing the processes reflected by inverting those conditions. For the current software system, below mentioned comes under the part of process specification:

Precondition 1: Valid password for Encryption.

Postcondition 1: Encrypt File.

Precondition 2: Invalid password for Encryption.

Postcondition 2: Generate error message.

Precondition 3: Valid password for Decryption.

Postcondition 3: Decrypt File.

Precondition 4: Invalid password for Decryption.

Postcondition 4: Generate error message.

Precondition 5: Invalid key length for encryption and decryption.

Postcondition 4: Generate error message.

CHAPTER-VI

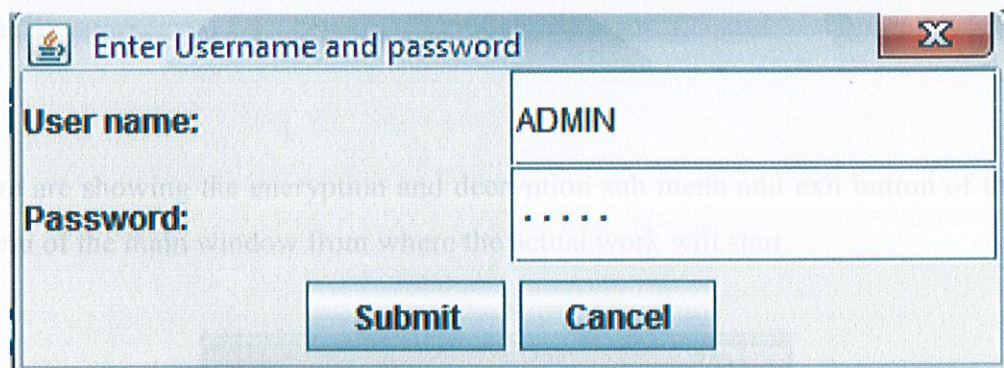
DEVELOPMENT

The algorithm used in our system for encryption and decryption of files is Advance Encryption Standard (AES). In this chapter we will discuss mainly the user interface design and the modular design of our system.

6.1 User Interface Design

It was required that we should have a user friendly interface so that a user has least problems in using the system for encryption and decryption, so after in depth analysis and design primitives created the following user interface designs which are evidently the snapshots of our system.

Authorization Login

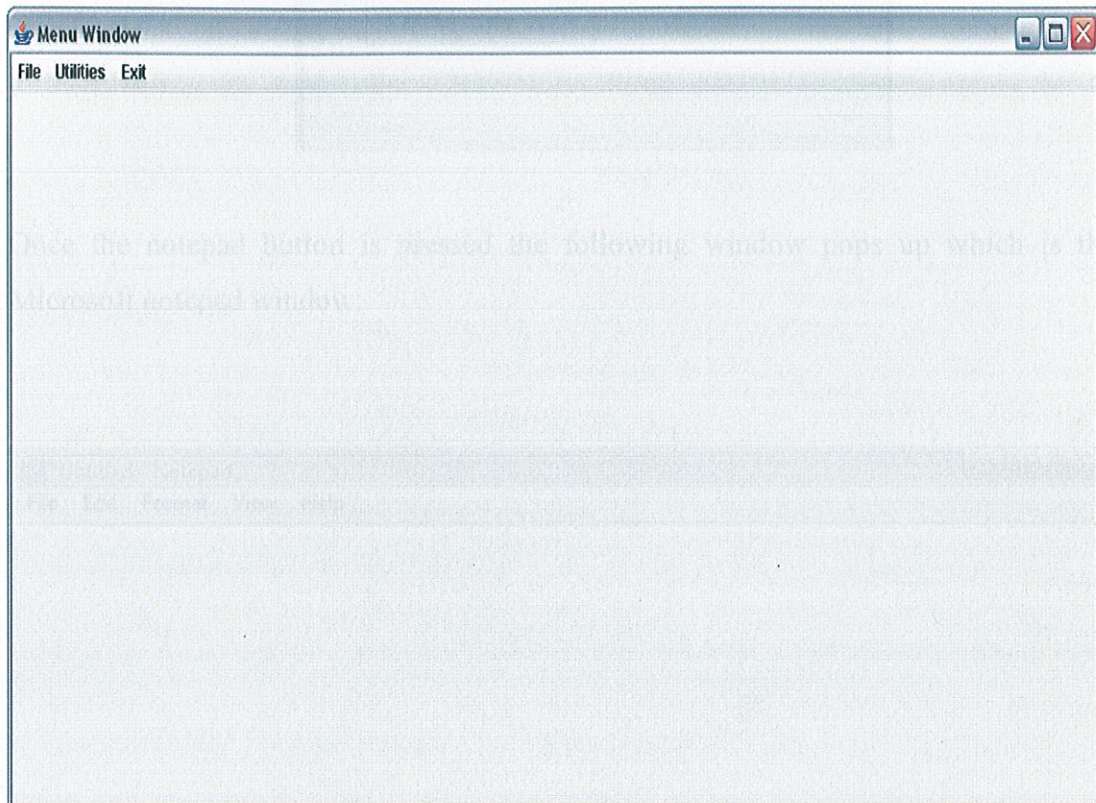


The image shows a Windows-style dialog box titled "Enter Username and password". It has a standard Windows icon in the top-left corner and a close button (X) in the top-right corner. The dialog contains two input fields: "User name:" with the text "ADMIN" entered, and "Password:" with seven dots entered. Below the input fields are two buttons: "Submit" and "Cancel".

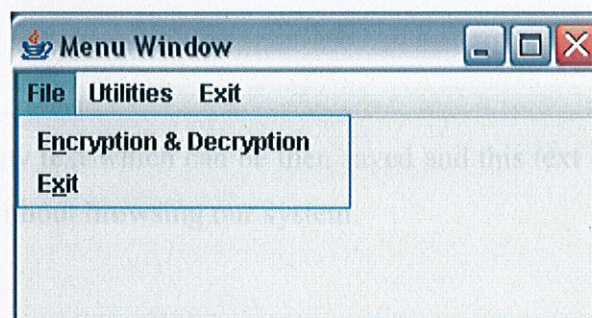
When we enter the correct user name and password which is with the administrator of this system, the validation of the user name and password entered by an user is done. After validating if the user name and password is entered is correct the user gets on the access to the main window form where the actual work will start.

Toolbar

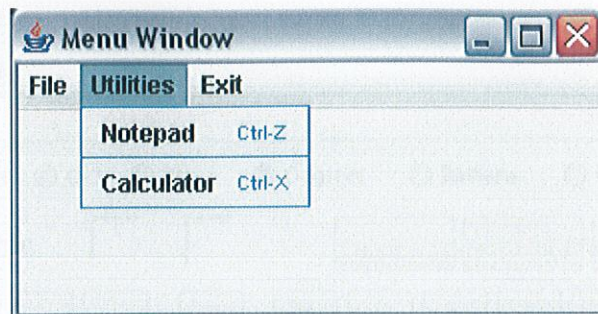
The following snap shots are the main window from where the actual work will start:



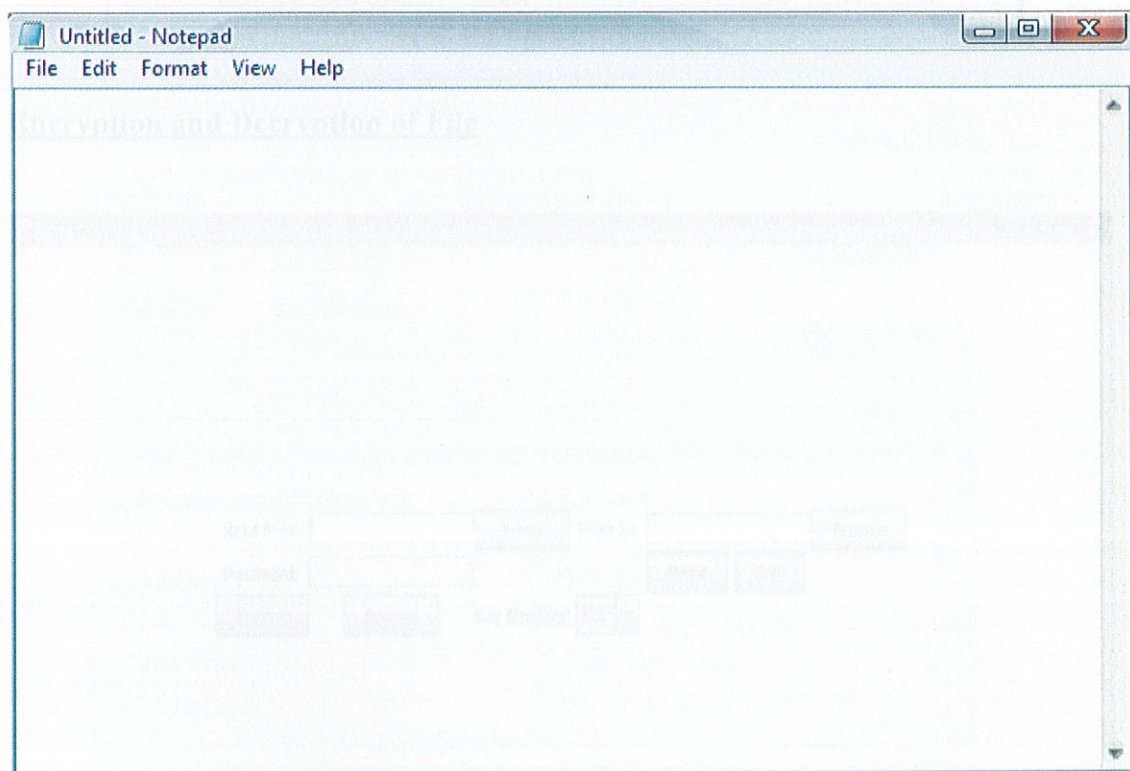
Here we are showing the encryption and decryption sub menu and exit button of the file menu of the main window from where the actual work will start.



This is the utility menu where we have notepad and calculator to help the user for any assistance with notepad and calculator.

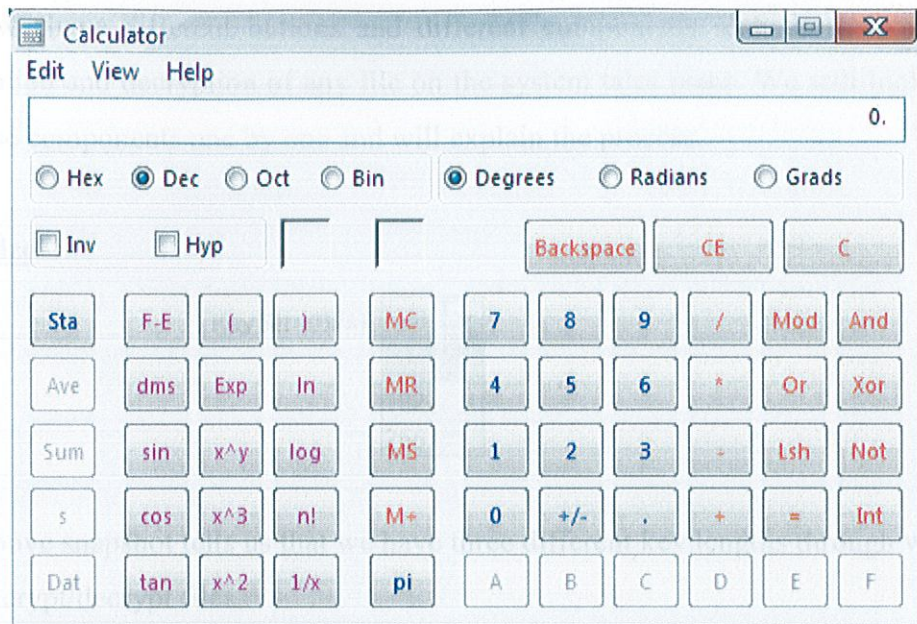


Once the notepad button is pressed the following window pops up which is the Microsoft notepad window:

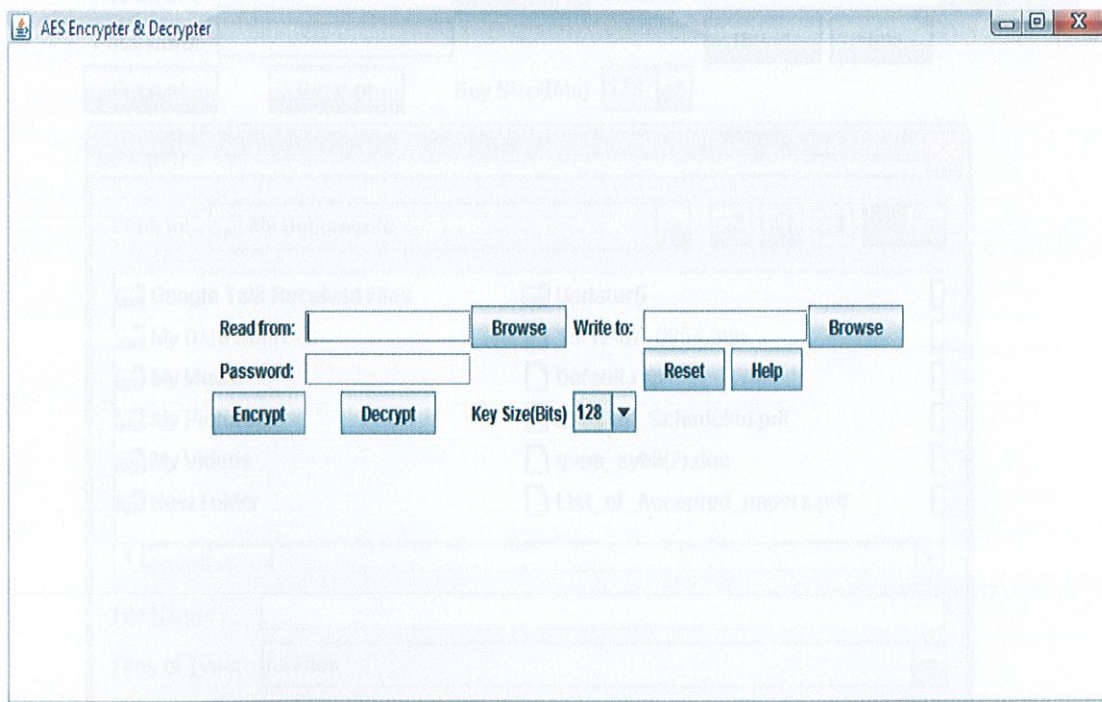


Here user can write any text which can be then saved and this text document can then be encrypted easily without browsing our system.

As we press the calculator button in the sub menu of utility a scientific calculator will pop up where user can calculate anything needed by him/her.



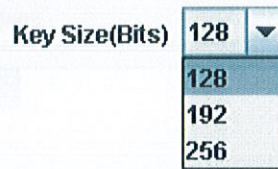
Encryption and Decryption of File



The above screen shot is the main form or the main window from where the user gets the interface for encryption and decryption of any file on the system.

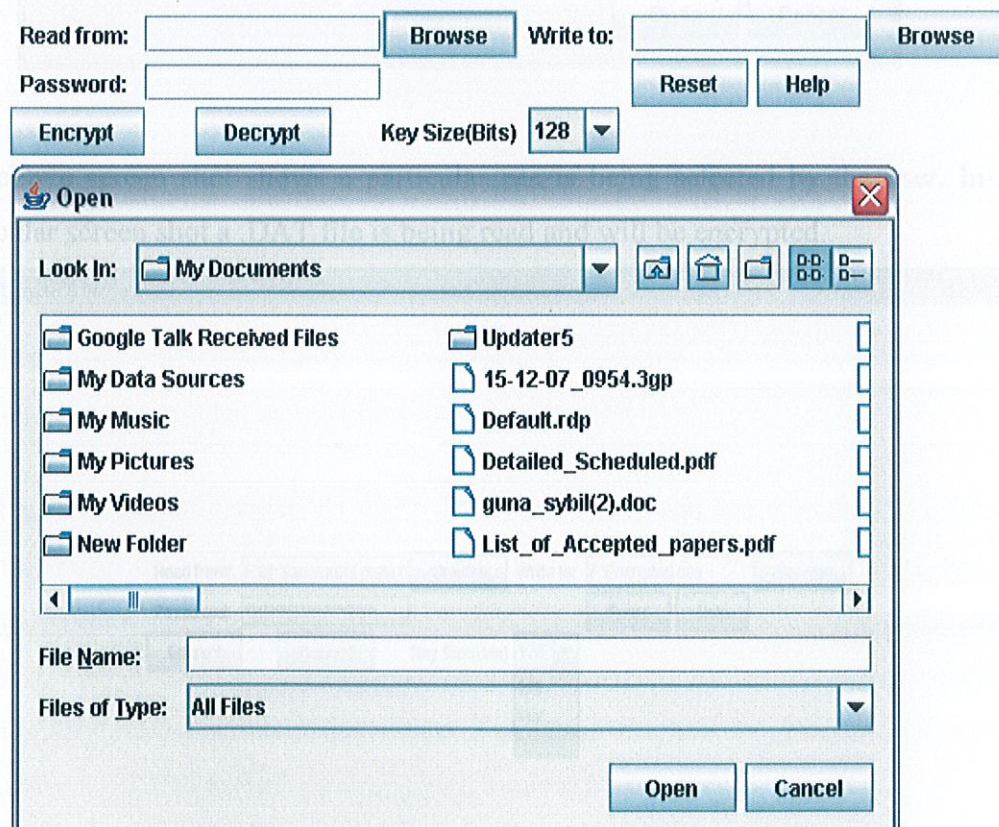
Here we have different buttons and different components with the help of which encryption and decryption of any file on the system takes place. We will look at each of these components one by one and will explain the process.

Key Size

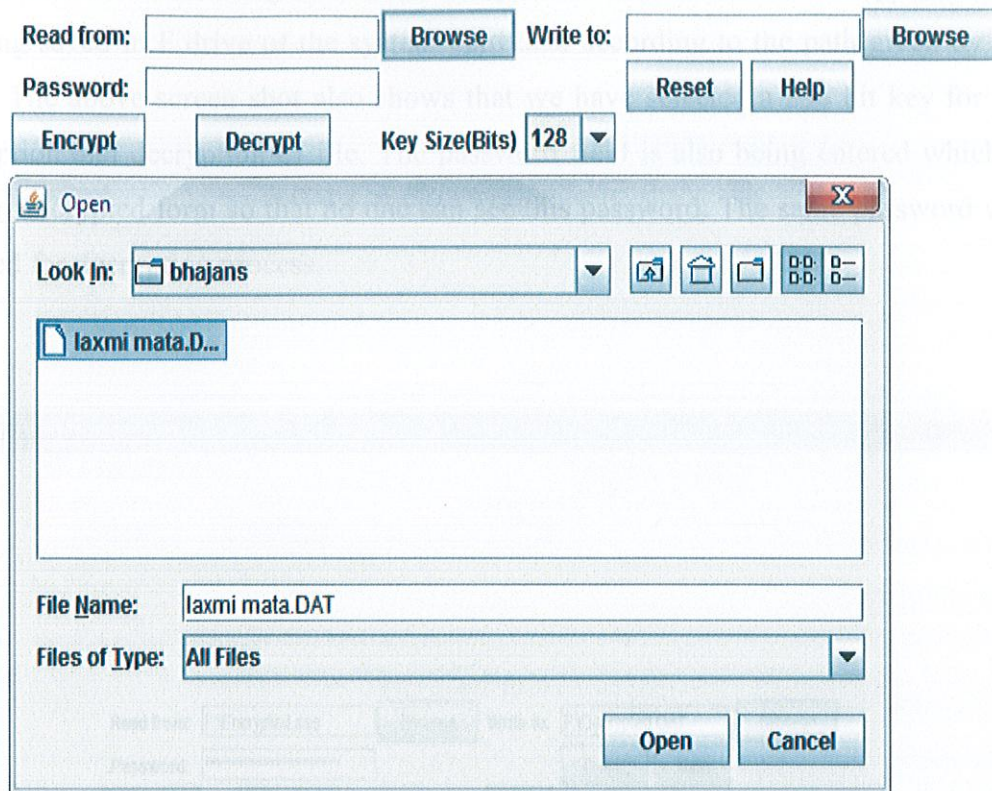


The above snapshot tells us that we have three different key lengths through which we can encrypt/decrypt files.

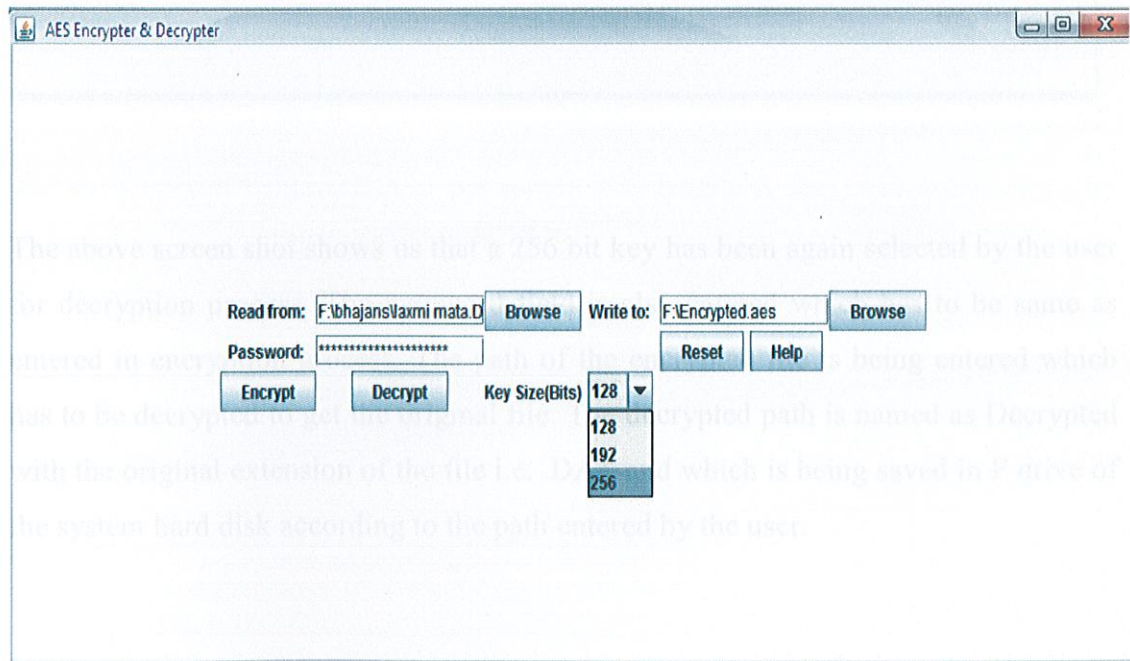
Encryption and Decryption of file :



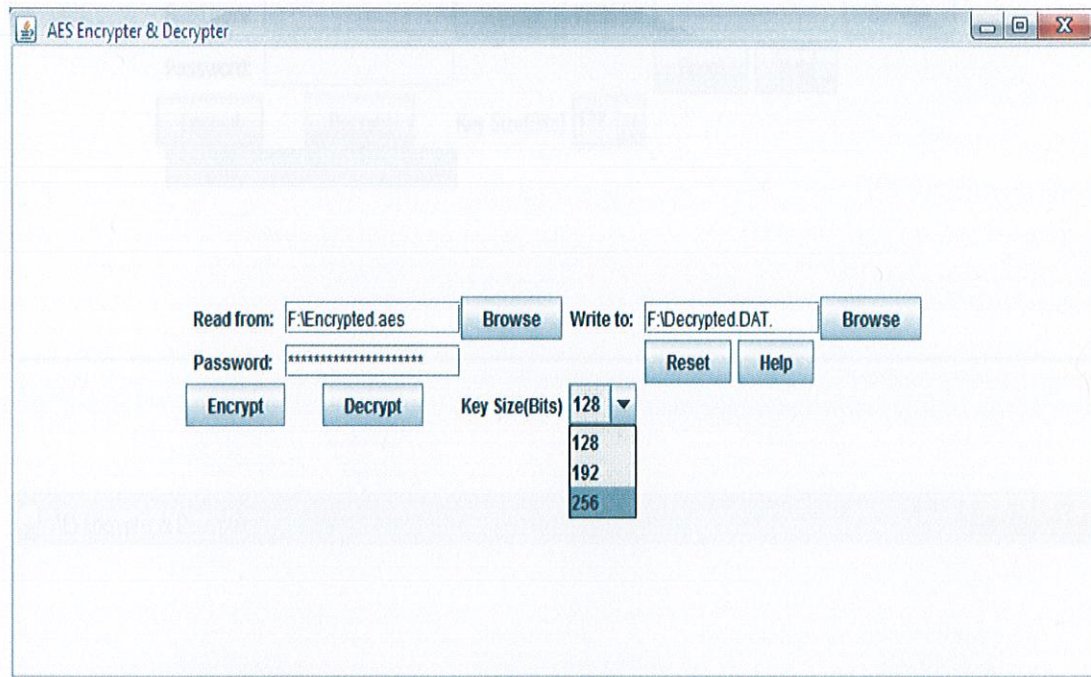
The above screen shot tells us that as we click the browse button the system files would be easily accessible to the user. These system files will be then encrypted or decrypted.



The above screen shot shows a particular file is being selected by the user. In this particular screen shot a .DAT file is being read and will be encrypted.

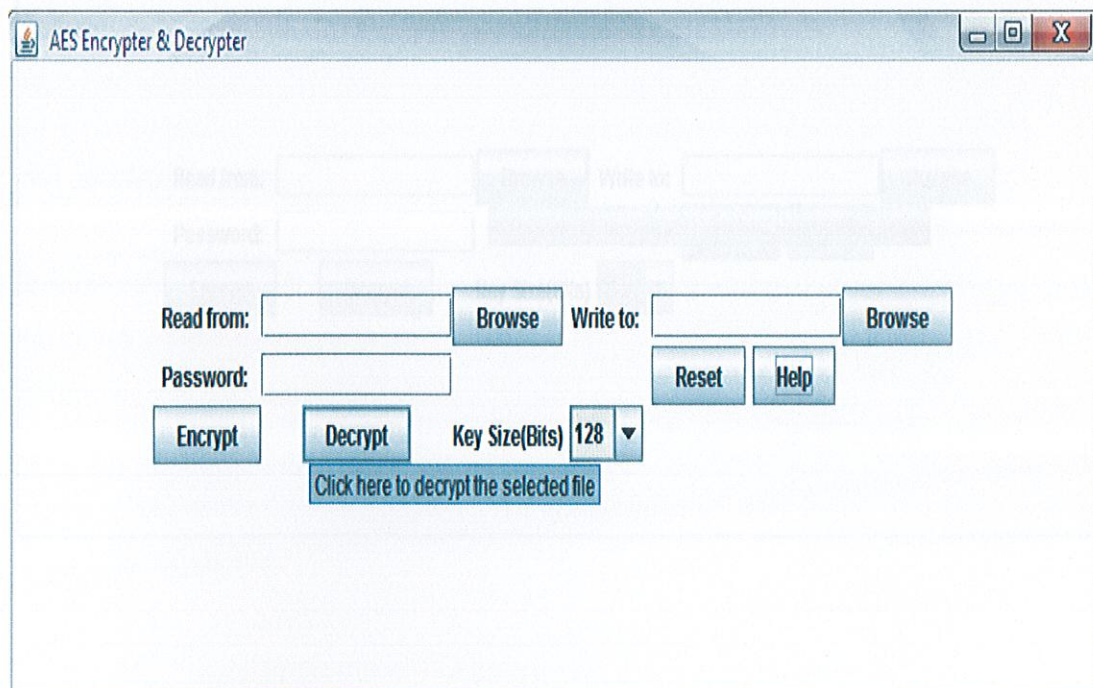
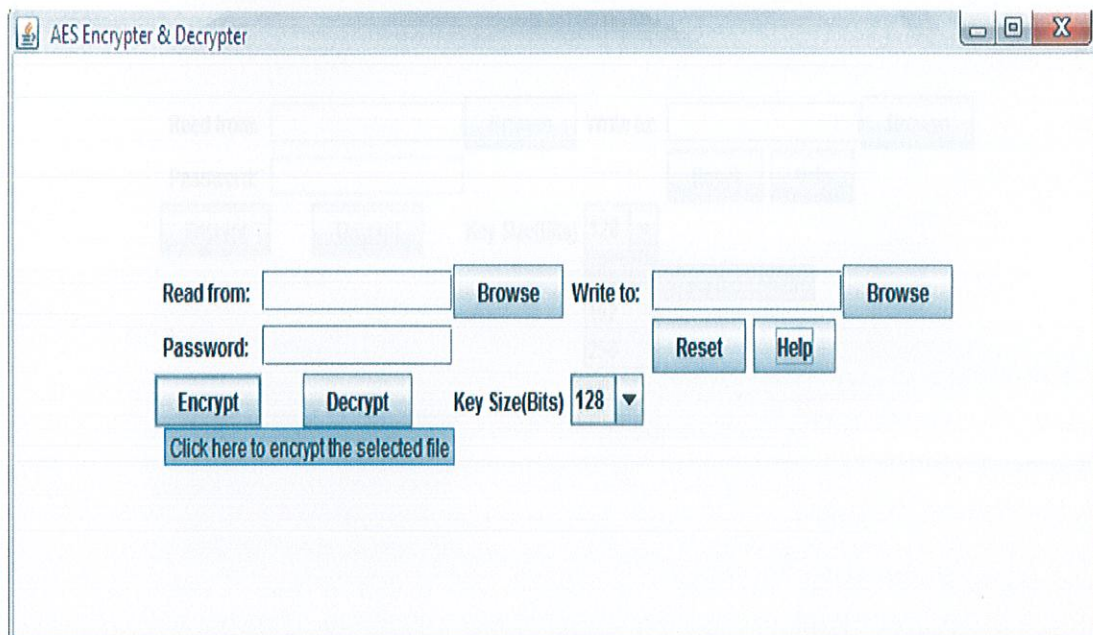


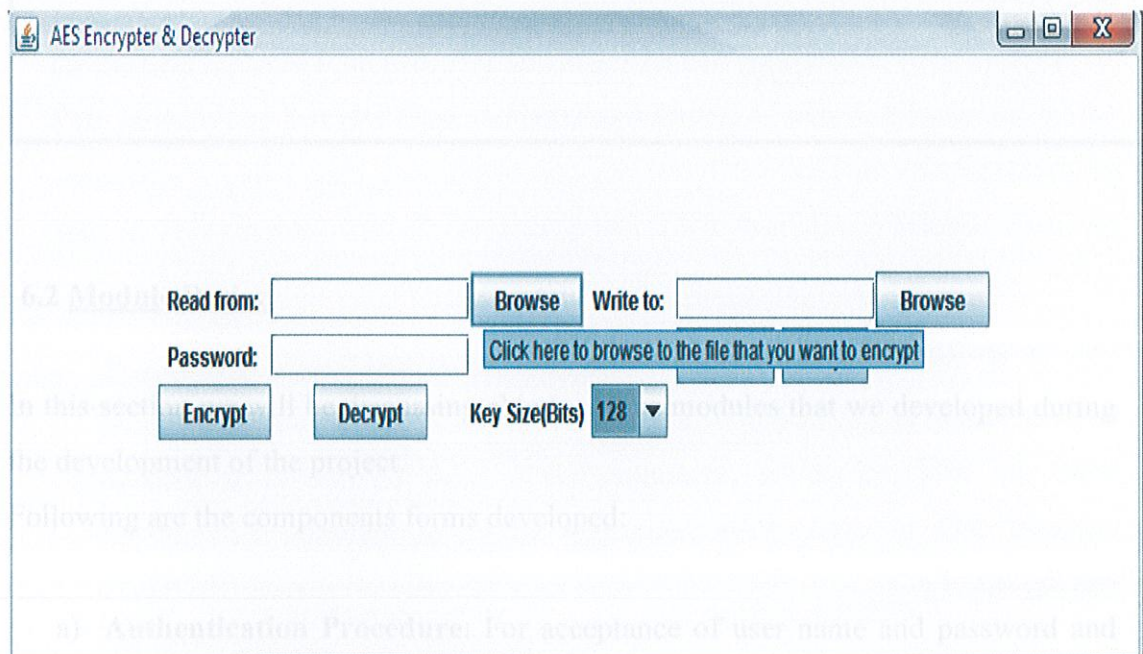
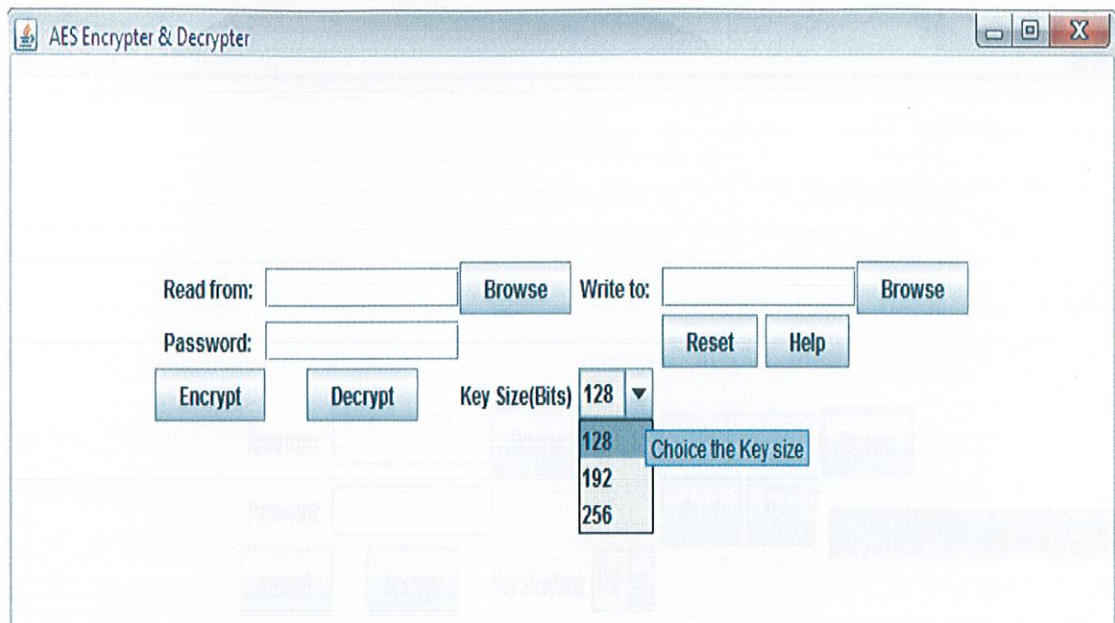
The .DAT file is then being read and is encrypted to .aes file named Encrypted which is being saved in F drive of the system hard disk according to the path given by the user. The above screen shot also shows that we have selected a 256 bit key for the encryption and decryption of file. The password field is also being entered which is also in encrypted form so that no one can see this password. The same password will be used for decryption process.

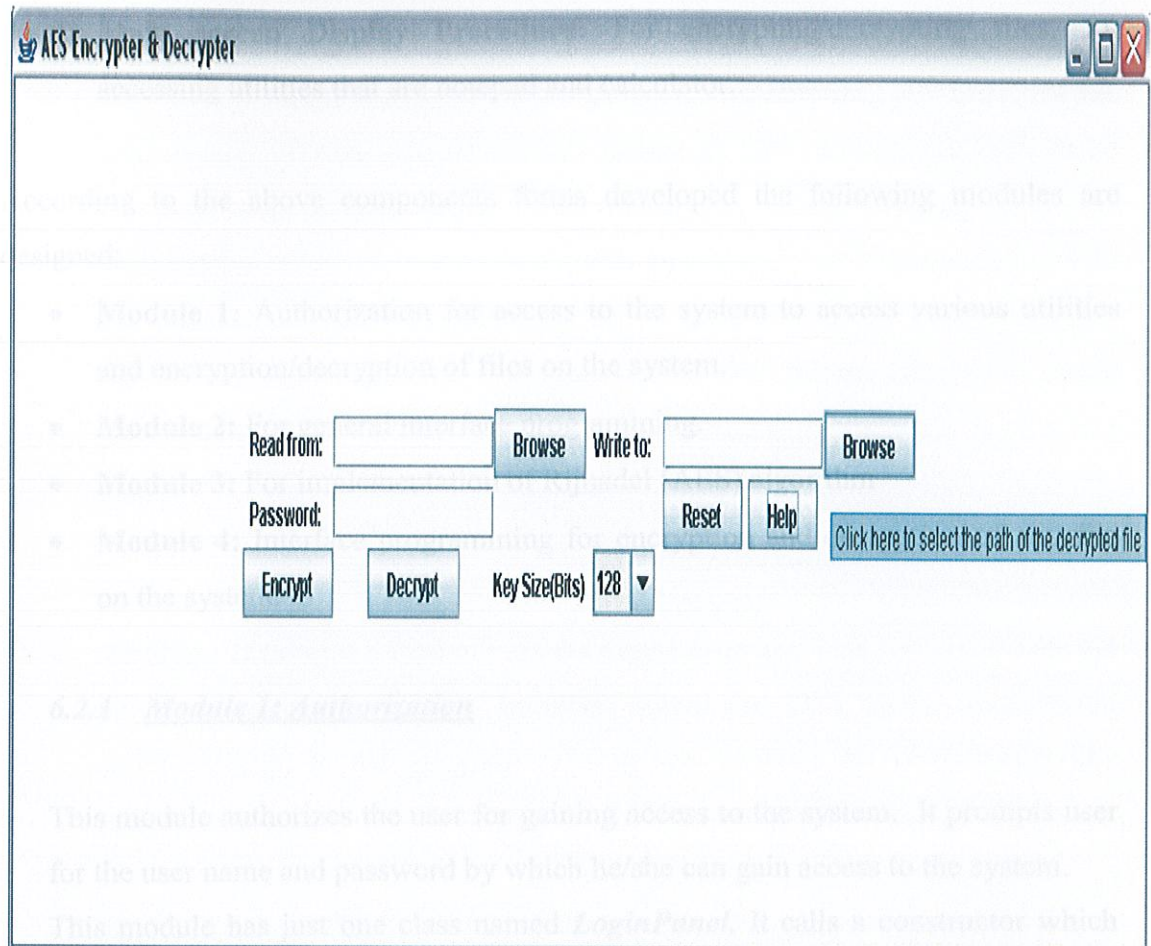


The above screen shot shows us that a 256 bit key has been again selected by the user for decryption process. The password field is also entered which has to be same as entered in encryption process. The path of the encrypted file is being entered which has to be decrypted to get the original file. The decrypted path is named as Decrypted with the original extension of the file i.e. .DAT and which is being saved in F drive of the system hard disk according to the path entered by the user.

Finally, following are some screen shot which shows the assistance given to the user for each of the functionalities in our encryption and decryption process:







6.2 Module Design

In this section we will be discussing about various modules that we developed during the development of the project.

Following are the components forms developed:

- a) **Authentication Procedure:** For acceptance of user name and password and its validation.
- b) **Algorithm Procedure:** For encryption & decryption process.
- c) **Main authorization login Procedure:** For gaining access to main window.

- d) **Main screen Display Procedure:** For encrypting/decrypting files, for accessing utilities that are notepad and calculator.

According to the above components forms developed the following modules are designed:

- **Module 1:** Authorization for access to the system to access various utilities and encryption/decryption of files on the system.
- **Module 2:** For general interface programming.
- **Module 3:** For implementation of Rijndael (AES) algorithm
- **Module 4:** Interface programming for encryption and decryption of any file on the system.

6.2.1 Module 1: Authorization

This module authorizes the user for gaining access to the system. It prompts user for the user name and password by which he/she can gain access to the system.

This module has just one class named **LoginPanel**, It calls a constructor which constructs a panel with user name and password fields. This class extends to **Jpanel**. This class consists of many components which are listed below:

- **TextField:** TextField is the simplest Swing text component. It is also probably its most widely used text component. JField allows us to edit one line of text. It is derived from JTextComponent, which provides the basic functionality common to swing text component. Here in Our program JTextField is used for enter the user name for the authentication to encrypt any file in the system.
- **JPasswordField:** JPasswordField is also swing component. JPasswordField allows us to edit one line of Password. JPasswordField is used for enter the password so that user/administrator authorized to use encryption and decryption program for encrypt/decrypt any file in the system.

- **JButton** : JButton is also swing component. The JButton class provides the functionality of push button. JButton allows an icon, a string, or both to be associated with the push button. Here we are using string to associate with the push button. When the button is pressed, an ActionEvent is generated. Using the ActionEvent object passed to the actionPerformed() method of the registration ActionListener, we can obtain the action command string associated with the button.

- **JDialog** : JDialog is a type of container used in swing. This container does not inherit JComponent. This does, however, inherit the AWT classes Component and Container. In our program JDialog box contains six components like submit button, username field, password field, cancel button etc.

- **ActionListener**: This is an interface that defines the actionPerformed() method that is invoked when an action event occurs. Listeners are created by implementing one or more of the interfaces defined by the Java.awt.event package. ActionListener is defined one method to receive action events.

Here is the authorization module procedure:

```
public class LoginPanel extends JPanel {
    private static Dimension dimension;
    private JTextField username;
    private JPasswordField password;
    private JButton okButton;
    private boolean ok;
    private JDialog dialog;
```

```
    public LoginPanel() {
        setLayout(new BorderLayout());
```



```

// construct a panel with user name and password fields
JPanel panel = new JPanel();
panel.setLayout(new GridLayout(2, 2));
    panel.setOpaque(false);
JLabel label = new JLabel("User name:");
label.setOpaque(false);
panel.add(label);
panel.add(username = new JTextField("Enter Username"));
panel.add(new JLabel("Password:"));
panel.add(password = new JPasswordField(""));
//password.setEcho('*');
add(panel, BorderLayout.CENTER);
//panel.setBackground( Color.green );

// create Ok and Cancel buttons that terminate the dialog
okButton = new JButton("Submit");
okButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent event) {
        ok = true;
        dialog.setVisible(true);
    }
});

JButton cancelButton = new JButton("Cancel");
cancelButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent event) {
        dialog.setVisible(true);
    }
});

// add buttons to southern border
JPanel buttonPanel = new JPanel();

```

```

        buttonPanel.add(okButton);
        buttonPanel.add(cancelButton);
        add(buttonPanel, BorderLayout.SOUTH);
    }

    public void setUser(String u) {
        username.setText(u);
    }

    public String getUser() {
        return "User";
    }

    public boolean showDialog(Component parent, String title) {
        ok = false;

        // locate the owner frame
        Frame owner = null;

        if (parent instanceof Frame) {
            owner = (Frame) parent;
        } else {
            owner = (Frame) SwingUtilities.getAncestorOfClass(Frame.class, parent);
        }

        // if first time, or if owner has changed, make new dialog
        if ((dialog == null) || (dialog.getOwner() != owner)) {
            dialog = new JDialog(owner, true);

            dialog.getRootPane().add(this);
            dialog.getRootPane().setDefaultButton(okButton);
            dialog.pack();
        }
    }

```

```

// set title and show dialog
dialog.setTitle(title);

Dimension d = Toolkit.getDefaultToolkit().getScreenSize();
Dimension frameSize = dialog.getSize();
dialog.setLocation((d.width - frameSize.width)/2, (d.height -
frameSize.height)/2); // center the mofo
dialog.setResizable(true);
dialog.setContentPane(this);
dialog.setVisible(true);

return ok;
}

public static void main(String[] args) throws IOException {
    LoginPanel pane = new LoginPanel();

    try {
        BufferedImage image = ImageIO.read(new File("mo1z.jpg"));
        pane.setBorder(new BackgroundImageBorder(image));
    } catch (IOException e) {
        e.printStackTrace();
    }

    //pane.setBorder(BorderFactory.createTitledBorder("A border"));
    pane.showDialog(null, "Enter Username and password");
}
}

```


6.2.2 Module 2: General Interface

This module create graphical user interface for the user for gaining access to the system. It prompts user for the file, utilities, and exit by which he/she can gain access to the system and system features.

This module has just one class named JMenubar, This class only defines the default constructor. It calls a constructor which constructs a menu with file, utilities and exit fields.

- **JMenu:** A top level window can have a menu bar associated with it. A menu bar displays a list of top-level menu choices. Each choice is associated with a drop-down menu. JMenu is also a container which is using swing operation in our program. components used in JMenu is file, utilities and exit in our program
- **ActionListener:** This is a interface define the action perform () method that is invoked when an action event occurs. Listeners are created by implementing one or more of the interfaces defined by the Java.awt.event package. ActionListener is defined one method to receive action events.

Here is the general interface module procedure:

```
public class GeneralUserInterface extends JMenuBar {
```

```
    String[ ] fileItems = new String[ ] { "Encryption & Decryption", "Exit" };
```

```
    String[ ] editItems = new String[ ] { "Notepad", "Calculator" };
```

```
    char[ ] fileShortcuts = { 'N', 'X' };
```

```
    char[ ] editShortcuts = { 'Z', 'X' };
```

```
    private Image image;
```

```
    public GeneralUserInterface()
```

```
    {
```

```

        super("");
    try {
        MediaTracker mt = new MediaTracker (this);
        image = Toolkit.getDefaultToolkit().getImage("untitled.jpg");
        mt.addImage(image, 0);
        mt.waitForID(0);
    }
    catch (Exception e) {
        e.printStackTrace();
    }

    JMenu fileMenu = fileMenu = new JMenu("File");
    JMenu editMenu = new JMenu("Utilities");
    JMenu otherMenu = new JMenu("Exit");

    ActionListener printListener = new ActionListener( ) {
        public void actionPerformed(ActionEvent event) {
            System.out.println("Menu item [" + event.getActionCommand( ) +
                               "] was pressed.");
        }
    };

    for (int i=0; i < fileItems.length; i++) {
        JMenuItem item = new JMenuItem(fileItems[i], fileShortcuts[i]);
        item.addActionListener(printListener);
        fileMenu.add(item);
    }

    // Assemble the File menus with keyboard accelerators.
    for (int i=0; i < editItems.length; i++) {
        JMenuItem item = new JMenuItem(editItems[i]);
        item.setAccelerator(KeyStroke.getKeyStroke(editShortcuts[i],
            Toolkit.getDefaultToolkit( ).getMenuShortcutKeyMask( ), false));
        item.addActionListener(printListener);
    }

```

```

        editMenu.add(item);
    }

    // Insert a separator in the Edit menu in Position 1 after "Undo".
    editMenu.insertSeparator(1);

    // Assemble the submenus of the Other menu.
    JMenuItem item;
    subMenu2.add(item = new JMenuItem("Extra 2"));
    item.addActionListener(printListener);
    subMenu.add(item = new JMenuItem("Extra 1"));
    item.addActionListener(printListener);
    subMenu.add(subMenu2);

    // Assemble the Other menu itself.
    otherMenu.add(subMenu);
    otherMenu.add(item = new JCheckBoxMenuItem("Check Me"));
    item.addActionListener(printListener);
    otherMenu.addSeparator( );
    ButtonGroup buttonGroup = new ButtonGroup( );
    otherMenu.add(item = new JRadioButtonMenuItem("Radio 1"));
    item.addActionListener(printListener);
    buttonGroup.add(item);
    otherMenu.add(item = new JRadioButtonMenuItem("Radio 2"));
    item.addActionListener(printListener);
    buttonGroup.add(item);
    otherMenu.addSeparator( );
    otherMenu.add(item = new JMenuItem("Potted Plant",
        new ImageIcon("image.gif")));
    item.addActionListener(printListener);

    // Finally, add all the menus to the menu bar.
    add(fileMenu);

```



```

        add(editMenu);
        add(otherMenu);
    }

    public static void main(String s[ ]) {
        public Image image;
        JFrame frame = new JFrame("Menu Window");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setJMenuBar(new GeneralUserInterface( ));
        frame.pack( );
        frame.setVisible(true);

        pane.setBorder(BorderFactory.createTitledBorder("A border"));
        pane.showDialog(null, "LoginPanel | Unit Test");
    }
}

```

6.2.3 Module 3: Rijndael (AES) algorithm

Here is the implementation procedure of Rijndael (AES) algorithm :

```

public class AESEncrypter
{
    Cipher ecipher;
    Cipher dcipher;

    public AESEncrypter(SecretKey key)
    {
        // Create an 8-byte initialization vector
        byte[] iv = new byte[]
        {
            0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,
            0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f
        }
    }
}

```

```

};

AlgorithmParameterSpec paramSpec = new IvParameterSpec(iv);
try
{
    ecipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
    dcipher = Cipher.getInstance("AES/CBC/PKCS5Padding");

    // CBC requires an initialization vector
    ecipher.init(Cipher.ENCRYPT_MODE, key, paramSpec);
    dcipher.init(Cipher.DECRYPT_MODE, key, paramSpec);
}
catch (Exception e)
{
    e.printStackTrace();
}

// Buffer used to transport the bytes from one stream to another
byte[] buf = new byte[1024];

public void encrypt(InputStream in, OutputStream out)
{
    try
    {
        // Bytes written to out will be encrypted
        out = new CipherOutputStream(out, ecipher);

        // Read in the cleartext bytes and write to out to encrypt
        int numRead = 0;
        while ((numRead = in.read(buf)) >= 0)
        {
            out.write(buf, 0, numRead);
        }
    }
}

```

```

        //System.out.println(numRead);
    }
    out.close();
}
catch (java.io.IOException e)
{
}
}

```

```

public void decrypt(InputStream in, OutputStream out)

```

```

{
    try
    {
        // Bytes read from in will be decrypted
        in = new CipherInputStream(in, dcipher);

        // Read in the decrypted bytes and write the cleartext to out
        int numRead = 0;
        while ((numRead = in.read(buf)) >= 0)
        {
            System.out.println(numRead);
            out.write(buf, 0, numRead);
        }
        out.close();
    }

    catch (java.io.IOException e)
    {
    }

    // System.out.println("end");
}

```



```

public static void main(String args[])
{
    System.out.printf("\t\t\tFinal Year Project \n\n ");

    System.out.println("\t\t\tProject Title: Encryption and ");
    System.out.println("\t\t\tDecryption of Files. \n\n");
    System.out.println("\t\t\tSubmitted By:- ");

    System.out.println("\t\t\tName: Kanishka Kumar Roll no: 041097 ");

    System.out.println("\t\t\tName: Pankaj Tripathi Roll no: 041010
\n\n");

    try
    {
        String temp,temp1,key1;

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

        System.out.print("Enter the path of file to be encrypted ");
        temp = br.readLine();
        int t= temp.lastIndexOf(".");
        StringBuffer sb = new StringBuffer(temp);
        String result = sb.substring(t);
        System.out.print("Enter the path where you want to store the
Decrypted file ");
        temp1 = br.readLine();
        // key1 = br.readLine();
        String sbg=temp1+"Decrypted"+result;
        //sb.replace(t,t+1,"new.");

        //String sbg = new String(sb);
    }
}

```

```

        System.out.println("The output File name is "+sbg);
        //String temp = "DESTest.dat";
        // Generate a temporary key. In practice, you would save this
key.
        // See also e464 Encrypting with DES Using a Pass Phrase.

        KeyGenerator kgen =
KeyGenerator.getInstance("AES");
        kgen.init(128);
        SecretKey key = kgen.generateKey();

        // Create encrypter/decrypter class
        AESEncrypter encrypter = new AESEncrypter(key);

        // Encrypt
        encrypter.encrypt(new FileInputStream(temp),new
FileOutputStream("Encrypted.txt"));
        // Decrypt
        encrypter.decrypt(new FileInputStream("Encrypted.txt"),new
FileOutputStream(temp1));
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
}
}

```

6.2.4 Module 4: Encryption/Decryption Interface

- **Container:** The container class is a subclass of Component. It has additional methods that allow other Component objects nested within it. Container objects can be stored inside a Container. A container is responsible for laying out any components that it contains. It does this through the use of various layout managers.
- **JLabel:** JLabel is the swing component that creates a label, which is a component that displays information. The label is swing's simplest component because it is passive. That is, a label does not respond to user input, it just display output.
- **JPasswordField:** JPasswordField is also swing component. JPasswordField allows us to edit one line of Password. JPasswordField is used for enter the password so that user/administrator authorized to use encryption and decryption program for encrypt/decrypt any file in the system.
- **JButton:** JButton is also swing component. The JButton class provides the functionality of push button. JBotton allows an icon, a string, or both to be associated with the push button. Here we are using string to associate with the push button. When the button is pressed, an ActionEvent is generated. Using the ActionEvent object passed to the actionPerformed() methed of the registration ActionListener, we can obtain the action commond string associated with the button.
- **GridBag Layout:** The general procedure for using a grid bag is to first create a new GridBagLayout object and make it the current layout manager. GridBagLayout defines only one constructor.

- GridBag Constraints: The location and size of each component in a gridbag are determined by a set of constraints linked to it. The constraints are contained in an object of type GridBagConstraints. Constraints include the height and width of a cell, and the placement, its alignment, and its anchor point within the cell.

Here is the encryption/ decryption interface module procedure:

```

public static Encryptor obj;
    public Container container;
    public JLabel passwordLabel, inputPathLabel, outputPathLabel,
statusLabel, bitLabel;
    public JTextField inputPathField, outputPathField;
    public JPasswordField passwordField;
    public JButton openButton, saveButton, encryptButton, decryptButton,
helpButton, resetButton;
    public GridBagLayout layout;
    public GridBagConstraints constraints;
    public JComboBox number;

    public boolean encrypt;
    public boolean busy;
    public int code[];
    public int codeLength;
    public int currentIndex;
    public DecimalFormat decimalFormat;
    public byte byteData[];
    public long filePointer, lengthOfFile;
    public int dataSize;

    public RandomAccessFile inputFilePointer;
    private RandomAccessFile outputFilePointer;

```

```

public Encryptor()
{
    super("AES Encrypter & Decrypter");
    obj = this;
    container = getContentPane();
    container.setBackground( Color.white );
    layout = new GridBagLayout();
    container.setLayout( layout );
    constraints = new GridBagConstraints();

    passwordLabel = new JLabel( "Password: " );
    inputPathLabel = new JLabel( "Read from: " );
    outputPathLabel = new JLabel( "Write to: " );
    statusLabel = new JLabel( "Status: stopped" );
    passwordField = new JPasswordField( 12 );
    passwordField.setEchoChar( '*' );
    inputPathField = new JTextField( 12 );
    outputPathField = new JTextField( 12 );
    openButton = new JButton("Browse");
    saveButton = new JButton("Browse");
    encryptButton = new JButton("Encrypt");
    decryptButton = new JButton("Decrypt");
    helpButton = new JButton("Help");
    resetButton = new JButton("Reset");
    decimalFormat = new DecimalFormat( "00" );

    bitLabel = new JLabel("Key Size(Bits)  ");
    number = new JComboBox();
    number.addItem("128 ");
    number.addItem("192 ");
    number.addItem("256 ");

```

```

passwordLabel.setHorizontalAlignment( SwingConstants.LEFT );
inputPathLabel.setHorizontalAlignment( SwingConstants.LEFT );
outputPathLabel.setHorizontalAlignment( SwingConstants.LEFT );
statusLabel.setHorizontalAlignment( SwingConstants.LEFT );
    bitLabel.setHorizontalAlignment( SwingConstants.LEFT );
    /*
openButton.setBackground( Color.lightGray );
saveButton.setBackground( Color.lightGray );
resetButton.setBackground( Color.lightGray );
helpButton.setBackground( Color.green );
encryptButton.setBackground( Color.red );
decryptButton.setBackground( Color.red );
passwordLabel.setBackground( Color.white );
inputPathLabel.setBackground( Color.white );
outputPathLabel.setBackground( Color.white );
passwordLabel.setForeground( Color.red );
inputPathLabel.setForeground( Color.blue );
outputPathLabel.setForeground( Color.blue );
statusLabel.setBackground( Color.white );
statusLabel.setForeground( Color.red );
    */
    openButton.setToolTipText( "Click here to browse to the file that you want to
encrypt" );
    saveButton.setToolTipText( "Click here to select the path of the decrypted
file" );
    resetButton.setToolTipText( "Click here to erase file paths" );
    helpButton.setToolTipText( "Click here to view help texts" );
    encryptButton.setToolTipText( "Click here to encrypt the selected file" );
    decryptButton.setToolTipText( "Click here to decrypt the selected file" );
    inputPathField.setToolTipText( "Type a file path here" );
    outputPathField.setToolTipText( "Type a destination file path here" );

```



```
passwordField.setToolTipText( "<html>Type any text as a code or password  
for file altering.<p>Press ENTER to enter advanced integer ASCII code(for  
advanced users).<p>Be sure to remember your code.</html>" );
```

```
number.setToolTipText("Choice the Key size");
```

```
addComponent( inputPathLabel, 1, 1, 4, 1 );  
addComponent( inputPathField, 1, 4, 4, 1 );  
addComponent( openButton ,      1, 8, 1, 1);  
addComponent( outputPathLabel,1, 10, 3, 1);  
addComponent( outputPathField,1, 13, 4, 1 );  
addComponent( saveButton, 1, 17, 1, 1 );  
addComponent( passwordLabel , 2, 1, 3, 1);  
addComponent( passwordField, 2, 4, 4, 1 );  
addComponent( resetButton, 2, 13, 2, 1 );  
addComponent( helpButton, 2, 15, 2, 1 );  
addComponent( encryptButton, 3, 1, 3, 1 );  
addComponent( decryptButton, 3, 5, 3, 1 );  
//addComponent( statusLabel, 3, 11, 7, 1 );  
addComponent( bitLabel, 3, 8, 3, 1);  
addComponent( number, 3, 10, 4, 1);
```

```
openButton.addActionListener( this );  
saveButton.addActionListener( this );  
encryptButton.addActionListener( this );  
decryptButton.addActionListener( this );  
helpButton.addActionListener( this );  
resetButton.addActionListener( this );  
passwordField.addActionListener( this );
```

```
setDefaultCloseOperation( JFrame.DO_NOTHING_ON_CLOSE );  
this.addWindowListener(  
    new WindowAdapter()  
{
```

```

public void windowClosing(WindowEvent e)
{
    int result = JOptionPane.showConfirmDialog( Encryptor.this, "Do you
really Want to exit?", "Confirmation", JOptionPane.YES_NO_OPTION );
    if( result == JOptionPane.YES_OPTION )
    {
        System.exit( 0 );
    }
}

);
pack();
setSize( 700, 175 );
setLocation( 20, 150 );
setResizable( true );
setVisible( true );
}

public void addComponent( Component component, int row, int column, int
width, int height )
{
    constraints.weightx = 0;
    constraints.weighty = 0;
    constraints.gridx = column;
    constraints.gridy = row;
    constraints.gridwidth = width;
    constraints.gridheight = height;
    layout.setConstraints( component, constraints );
    container.add( component );
}

public void actionPerformed((ActionEvent event)
{
    if( event.getSource() == openButton )
    {

```

```

JFileChooser fileChooser = new JFileChooser();
fileChooser.setFileSelectionMode( JFileChooser.FILES_ONLY );
int result = fileChooser.showOpenDialog( this );
if( result != JFileChooser.APPROVE_OPTION )
    return;
File fileName = fileChooser.getSelectedFile();
if( fileName == null || fileName.getName().equals("") )
{
    JOptionPane.showMessageDialog( this, "Invalid file name" , "Error",
JOptionPane.ERROR_MESSAGE );
}
else
{
    inputPathField.setText( fileName.getPath() );
    if( outputPathField.getText().equals( "" ) )
    {
        outputPathField.setText( fileName.getPath() );
    }
}
else if( event.getSource() == saveButton )
{
    JFileChooser fileChooser = new JFileChooser();
    fileChooser.setFileSelectionMode( JFileChooser.FILES_ONLY );
    int result = fileChooser.showSaveDialog( this );
    if( result != JFileChooser.APPROVE_OPTION )
        return;
    File fileName = fileChooser.getSelectedFile();
    if( fileName == null || fileName.getName().equals("") )
    {
        JOptionPane.showMessageDialog(this, "Invalid file name" , "Error",
JOptionPane.ERROR_MESSAGE );
    }
}

```



```

else
{
    outputPathField.setText( fileName.getPath() );
}
}
else if( event.getSource() == encryptButton || event.getSource() ==
decryptButton )
{
    Worker worker = new Worker( event, container );
    worker.start();
}
else if( event.getSource() == helpButton )
{

```

String tmp = "File Encryptor and Decryptor\n Designed by Kanishk Kumar and Pankaj Tripathi \nE-mail: 'kanishka.kr@gmail.com and pankaj.tripathi.juit@gmail.com\n\nThis program is used to make a file(encrypt a file) such that \nNO ONE CAN READ IT.\nType file Paths and Type character codes\nand click ENCRYPT or DECRYPT Button to perform file operation.\nIf you encrypt a file with a code you can retrieve original file by\ndecrypting it with only the previously applied code used for encryption.\n\n\n" +

"A simple example:\nClick RESET.\nClick the left blue 'Browse' marked button.\nSelect any file\n(do not select important file yet, as you are practicing).\nThen type a password or code at the code text field.\nClick ENCRYPT button.\nAt this step your file is encrypted.\nJust click the DECRYPT button again to decrypt the file to\nretain to its original contents.\nThis is a simple example but\nyou can change the path of 'Write to' text field to another file\nfor keeping the source file unchanged(specially for safety, but requires much space).\n\n\nWarning:\nDo not encrypt a file more than once.\nAlways make sure that you DO NOT FORGET THE CODE." +

"\n\n\nNote:\nEncrypt a file with your secret code to make it unreadable\nand when you want to read that file decrypt it with the same code to make it readable.\nYou can use this program to any kind of file such as text file, execution file etc.\nThis program processes only one file at a time so\nif you want

to encrypt several files at a time convert them into a single file\n(for example making all the files as a zip file etc.)and encrypt that single file.\nThis program takes about 2 to 3 minutes to process a file of 500 MB length.\nNote that you cannot change a file which is set to 'Read-Only' mode by the operating system.\nUncheck the 'Read-only' attribute before proceeding." +

"\n\n\nExtra Notes(for advanced users only):\nEncrypt operation with a code is total inverse of\nDecrypt operation with the same code,\nso if you find any problem such as 'you encrypted twice' etc.\ntry doing the inverse operation the same number of time in inverse sequence.\n\n\nAdvanced code:\nYou can also enter code as ASCII integer values by pressing ENTER\nwhen the focus is on the code text field.\nEnter as many integers as you want but\nmake sure to write it down sequentially or remember its sequence.";

```

        JTextArea ta = new JTextArea( 10, 40 );
        ta.setText( tmp );
        ta.setEditable( false );
        ta.select( 0, 0 );
        JOptionPane.showMessageDialog( this, new JScrollPane( ta ),
"Help/About", JOptionPane.INFORMATION_MESSAGE );
    }
    else if( event.getSource() == resetButton )
    {
        inputPathField.setText( "" );
        outputPathField.setText( "" );
    }
    else if( event.getSource() == passwordField )
    {
        try
        {
            if( busy )
            return;
            busy = true;
            int rslt;
            int length=0;

```

```

String inputString;
int input;
byte byteTmp;
Vector byteCode = new Vector();
int count = 0;
while( true )
{
    rslt = JOptionPane.showConfirmDialog( this, "Proceed with Integer
number " + (count+1) + "?", "Proceed?", JOptionPane.YES_NO_OPTION);
    if( rslt != JOptionPane.YES_OPTION )
    {
        break;
    }
    inputString = JOptionPane.showInputDialog( this, "Enter an
Integer\n(Range: 0 - 255)" );
    input = Integer.parseInt( inputString );
    if( input < 0 || input > 255 )
    {
        break;
    }
    byteTmp = ( byte )input;
    byteCode.addElement( new Byte( byteTmp ) );
    length++;
    count++;
}
if( length > 0 )
{
    String output = "";
    for( count = 0; count < length; count++ )
    {
        output += ( char )( ( ( Byte ) byteCode.elementAt( count )
).byteValue() );
    }
}

```



```

        passwordField.setText( output );
    }
    busy = false;
}
catch( Exception exception )
{
    JOptionPane.showMessageDialog(this, exception.toString(), "Exception",
JOptionPane.ERROR_MESSAGE );
    busy = false;
}
}
}

```

```

public static void main( String args[] )

```

```

{
    Encryptor application = new Encryptor();
}

```

```

public class UpdateThread extends Thread

```

```

{
    Container container;

    public UpdateThread( JLabel a, int b, int c )
    {

```

```

        label = a;
        progress = b;
        total = c;
    }
    public void run()
    {

```

```

        int a;
        a = ( progress * 100 ) / total;
        String b = "Progress: ";

        b += decimalFormat.format( a );

        b += " %";
        label.setText( b );
    }
}

```

```

public class Worker extends Thread
{
    ActionEvent event;
    Container container;
    public Worker( ActionEvent a, Container b )
    {
        event = a;
        container = b;
    }
    public void run()

```

```

{
    try
    {
        KeyGenerator kgen =
KeyGenerator.getInstance("AES");
        kgen.init(128);
        SecretKey key = kgen.generateKey();
        AESEncrypter encrypter = new AESEncrypter(key);

        // Create encrypter/decrypter class

        busy = true;

        container.setCursor(
Cursor.getPredefinedCursor(
Cursor.WAIT_CURSOR ));

        openButton.setEnabled( false );
        saveButton.setEnabled( false );
        encryptButton.setEnabled( false );
        decryptButton.setEnabled( false );
        helpButton.setEnabled( false );
        resetButton.setEnabled( false );

        if( event.getSource() == encryptButton )
            encrypt = true;
        else
            encrypt = false;

        if( inputPathField.getText().trim().length() < 1 )
        {

```



```

        JOptionPane.showMessageDialog( Encryptor.this , "Enter an input file
name" , "Error", JOptionPane.ERROR_MESSAGE );
        return;
    }

    if( outputPathField.getText().trim().length() < 1 )
    {
        JOptionPane.showMessageDialog( Encryptor.this , "Enter a output file
name" , "Error", JOptionPane.ERROR_MESSAGE );
        return;
    }

    String tmpString = new String( passwordField.getPassword() );

    codeLength = tmpString.length();
    if( codeLength < 1 )
    {
        JOptionPane.showMessageDialog( Encryptor.this, "Please type a code.",
"Error", JOptionPane.ERROR_MESSAGE );
        return;
    }
    code = new int[ codeLength ];
    for( int count = 0; count < codeLength; count++ )
    {
        code[ count ] = ( ( int )( tmpString.charAt( count ) ) % 256 );
    }

    if( !encrypt )
    {
        try
        {
            String temp,sbg;
            openButton.setEnabled( false );

```

```

        temp=inputPathField.getText();
        sbg=outputPathField.getText();
        encrypter.encrypt(new FileInputStream(temp),new
FileOutputStream("c:/New Folder/Encrypted.txt"));
        encrypter.decrypt(new FileInputStream("c:/New
Folder/Encrypted.txt"),new FileOutputStream(sbg));

    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
}

System.err.println("Input File Not Readable");

if(encrypt){
    try
    {
        String temp,temp1,key1,sbg;

        temp=inputPathField.getText();
        sbg=outputPathField.getText();

        encrypter.encrypt(new FileInputStream(temp),new
FileOutputStream(sbg));
        encrypter.decrypt(new FileInputStream(sbg),new
FileOutputStream("c:/New Folder/Decrypted.txt"));

    }
    catch (Exception e)
    {

```

```

        e.printStackTrace();
    }
}

/* File file1 = new File( inputPathField.getText() );
File file2 = new File( outputPathField.getText() );
if( !file1.exists() || !file1.isFile() )
{
    JOptionPane.showMessageDialog( Encryptor.this, "Invalid input file
name" , "Error", JOptionPane.ERROR_MESSAGE );
    return;
}
else if( !file1.canRead() )
{
    System.err.println("Input File Not Readable");
    return;
}

if( file2.exists() && !file2.canWrite() )
{
    JOptionPane.showMessageDialog( Encryptor.this, "The selected output
file is read-only!\nPlease select another output file.", "Operation halted",
JOptionPane.ERROR_MESSAGE );
    return;
}

String str1 = "File Size " + file1.length() + " bytes\nDo you want to
proceed with this operation?";
if( file1.getPath().equals( file2.getPath() ) )
{
    openButton.setEnabled( true );
    saveButton.setEnabled( true );
}
}

```



```

        str1 += "\n\n( Warning : If a power failure occurs during this
process\nall file data to be corrupted and can't be retrieved anymore,\nbecause you
are going to overwrite existing file rather \ncreating a new file.)";

```

```

    }
    else
    {
        if( file2.exists() )
        {
            str1 += "\n\n( Warning : You are going to overwrite an existing file.)";
        }
    }

    int result = JOptionPane.showConfirmDialog( Encryptor.this, str1,
"Confirmation", JOptionPane.YES_NO_OPTION );
    if( result != JOptionPane.YES_OPTION )
    {
        return;
    }
*/
}

catch( Exception exception )
{
    JOptionPane.showMessageDialog( Encryptor.this, exception.toString(),
"Exception", JOptionPane.ERROR_MESSAGE );
}

finally
{
    try{
        code = null;
        busy = false;
        openButton.setEnabled( true );
        saveButton.setEnabled( true );

```

```

encryptButton.setEnabled( true );
decryptButton.setEnabled( true );
helpButton.setEnabled( true );
resetButton.setEnabled( true );
statusLabel.setText( "Status: stopped" );

try
{
    if( inputFilePointer != null )
    {
        inputFilePointer.close();
        inputFilePointer = null;
    }
    if( outputFilePointer != null )
    {
        outputFilePointer.close();
        outputFilePointer = null;
    }
    container.setCursor( Cursor.getPredefinedCursor(
Cursor.DEFAULT_CURSOR ) );
}
catch( Exception exception )
{
    JOptionPane.showMessageDialog( Encryptor.this, exception.toString(),
"Exception", JOptionPane.ERROR_MESSAGE );
}

catch(Exception E){
}
}
}
}

```

CHAPTER-VII

INTEGRATION AND TESTING

In this phase the different components developed in the implementation phase were integrated unto one single package and then tested for various parameters. Software testing is a critical element of software quality assurance and represents the ultimate review of specification, design, and code generation. Below mentioned are rules which serve as good testing objectives:

- a) Testing is a process of executing a program with the intent of finding an error.
- b) A good test case is one that has high probability of finding an as-yet-undiscovered error.
- c) A successful test is one that uncovers an as-yet-undiscovered error.

These objectives imply a dramatic change in viewpoint. They move counter to the commonly held view that a successful test is one in which no errors are found. Our objective is to design tests that systematically uncover different classes of errors and to do so with a minimum amount of time and effort.

7.1 Testing Principles

Before applying methods to design effective test cases, a software engineer must understand the basic principles that guide software testing. Following are some of them:

- a) **All tests should be traceable to customer requirements:** As we know, the objective of software testing is to uncover errors. It follows that the most severe defects (from the customer's point of view) are those that cause the program to fail to meet its requirements.
- b) **Tests should be planned long before testing begins:** Test planning can begin as soon as the requirements model is complete. Detailed definition can begin as soon as the design model has been solidified. Therefore, all tests can be planned and designed before any code has been generated.

- c) **The Pareto principle applies to the software testing:** Stated simply, the Pareto principle implies that 80 percent of all errors uncovered during testing will likely be traceable to 20 percent of all program components. The problem, of course, is to isolate these suspect components and to thoroughly test them.
- d) **Testing should begin “in the small” and in progress towards testing “in the large”:** The first tests planned and executed generally focus on individual components. As testing progresses, focus shifts in an attempt to find errors in integrated clusters of components and ultimately in the entire system.
- e) **Exhaustive testing is not possible:** the number of path permutations for even a moderately sized program is exceptionally large. For this reason, it is impossible to execute every combination of paths during testing. It is possible, however, to adequately cover program logic and to ensure that all conditions on the component-level design have been exercised.
- f) **To be most effective, testing should be conducted by an independent third party:** *By most effective*, we mean testing that has the highest probability of finding errors (the primary objective of testing).

Testing software is generally carried out in two stages:

- a) **Black Box Testing:** This form of testing is done on the system as a whole where we are not concerned with the system inside functionality: we just take an input and find out whether it maps to its correct output.
- b) **White Box Testing:** This form of testing takes into account the system inside functionally and it requires testing done at each particular level or module of the system.

7.2 Black Box testing

Black Box testing consists of the following four types:

- a) Graph Based Testing
- b) Equivalence Testing
- c) Boundary Value Analysis
- d) Comparison Testing

The type of black box testing our team did was Graph based testing, namely Cause-Effect Graphs:

In cause-effect graphs following points have to be kept in mind:

- We have to describe all causes (inputs) and effects (outputs).
- To number the causes and effects and make them node of a graph.
- Establish logical connections between the nodes.
- Supply additional information necessary.
- Develop test cases.

Black Box testing for encryption Process:

Requirement: the system should successfully encrypt plaintext or a file.

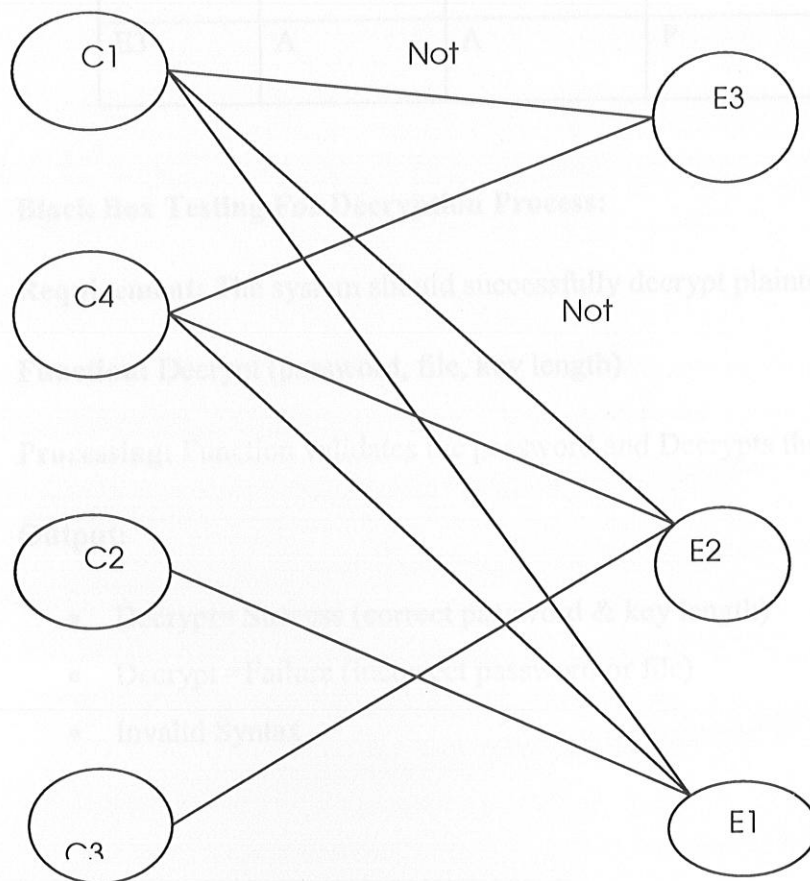
Function: Encrypt (Password, file, key length)

Processing: Function validates the password and encrypts the file

Output:

- Encrypt=Success (correct password & key length)
- Encrypt=Failure (incorrect password or file)
- Invalid Syntax

Causes	Effects
C1, The characters E,N,C,R,Y,P,T	E1, The message Encrypt=Success
C2, Calculate Encrypt=Success	E2, The message Encrypt=Failure
C3, Calculate Encrypt=Failure	E3, The message Invalid Syntax
C4, The two parameters	



Cause-Effect Graph

Building Test Case: For developing test cases the following assumptions are kept:

I: Input Present, S: Input Absent, X: Don't Care, P: Output Present, A: Output Absent

	TEST1	TEST2	TEST3	TEST4
C1	I	I	I	S
C2	I	S	X	X
C3	S	I	X	X
C4	I	I	S	I
E1	P	A	A	A
E2	A	P	A	A
E3	A	A	P	P

Black Box Testing For Decryption Process:

Requirement: The system should successfully decrypt plaintext or a file.

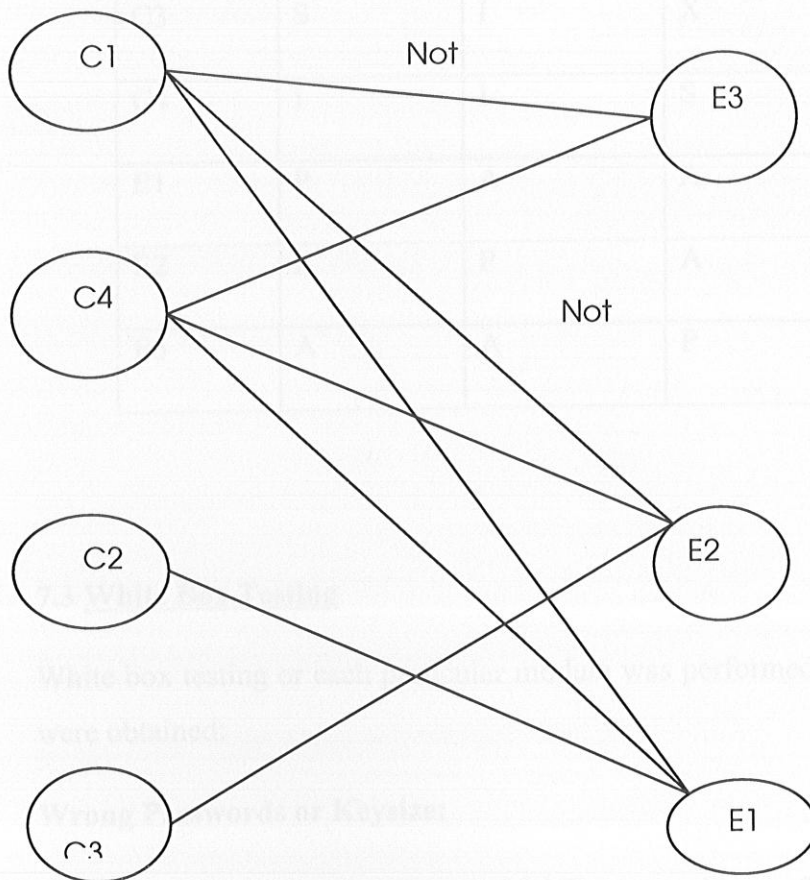
Function: Decrypt (password, file, key length)

Processing: Function validates the password and Decrypts the file.

Output:

- Decrypt= Success (correct password & key length)
- Decrypt =Failure (incorrect password or file)
- Invalid Syntax

Causes	Effects
C1. The characters D,E,C,R,Y,P,T	E1. The message Decrypt=Success
C2. Calculate Decrypt = Success	E2. The message Decrypt = Failure
C3. Calculate Decrypt = Failure	E3. The message Invalid Syntax
C4. The two parameters	



Cause-Effect Graph

Building Test Case: Following test cases are developed:

	TEST1	TEST2	TEST3	TEST4
C1	I	I	I	S
C2	I	S	X	X
C3	S	I	X	X
C4	I	I	S	I
E1	P	A	A	A
E2	A	P	A	A
E3	A	A	P	P

7.3 White Box Testing

White box testing of each particular module was performed and the following results were obtained:

Wrong Passwords or Keysize:

- If we enter wrong password while encrypting or decrypting files we get the following error message:
- If we provide with two different key sizes in encryption and decryption process then we receive the following error message:
- Password is case dependent i.e. both the keys in encryption and decryption should be either be in lower case or upper case otherwise the following error message is received:

The following formats of data the software system can encrypt/decrypt successfully

Text Files:

Notepad File: *.txt

Word File: *.doc

WordPad File: *.ppt

Access File: *.mdb

Excel File: *.xls

PowerPoint Template: *.pot

MHTML Document: *.mht

Document Template: *.dot

Firefox Document: *.htm

Picture Files:

Windows Meta File: *.wmf

Extended Windows Meta File: *.emf

Image Files: *.gif, *.png, *.tif, *.bmp, *.jpeg

Audio Files

MPEG-1 Audio Layer 3 File: *.mp3

Windows Media Audio File: *.wma

Ogg Vorbis Audio File: *.ogg

Waveform Audio Format: *.wav, *. vox

CHAPTER-VIII

INSTALLATION AND ACCEPTANCE

After the testing phase is over the software systems is installed on the system where its application is required. The acceptance of the software produces the following quality management protocols to be examined:

8.1 Quality Management

Achieving is a high level or product or service quality is the objective of most organizations. It is no longer acceptable to deliver poor quality products and then repair problems and deficiencies after they have been delivered to the customer. In this respect, software is the same as any manufactured product such as cars, television or computers.

Software quality management can be structured into three principal activities:

- a) *Quality Assurance*: The establishment of a framework of organizational procedures and standards which lead to high-quality software.
- b) *Quality Planning*: The selection of appropriate procedures and standards from this framework and the adaptation of these for a specific software product.
- c) *Quality control*: The definition and enactment of processes which ensure that the project quality procedures and standards are followed by the software development team.

Quality management provides an independent check on the software development process. The deliverables from the software process are input to the quality management process and are checked to ensure that they are consistent with organizational standards and goals. As the quality assurance and control team should be independent, they can take an objective view of the process and can report problems and difficulties to senior management in the organization.

Areas covered to model quality assurance:

Management Responsibility	Quality Team
Control of non conforming products	Design Control
Handling, Storage, Packaging & Delivery	Purchasing
Purchaser-supplied products	Identification & Traceability
Process Control	Inspection & test Status
Inspection & Test Equipment	Inspection & Test Status
Contract Review	Corrective Action
Document Control	Quality Records
Internal Quality Audits	Training
Servicing	Statistical Technique

8.2 Quality Assurance & standards

Quality assurance (QA) activities define a framework for achieving software quality. The QA process involves defining or selecting standards that should be applied to the software development process or software product. These standards may be embedded in procedures or processes which are applied during development.

There are two types of standards that may be established as part of quality assurance process:

- a) *Product standards*: These are standards that apply to the software product being developed. They include standards such as the structure of the requirements document which should be produced, documentation standard such as standard comment header for an object class definition & coding standards which define how a programming language should be used.
- b) *Process standards*: These are standards that define the processes which should be followed during software development. They may include definitions of specifications, design & validation processes and a description of the documents which must be generated in the course of these processes.

8.3 Product & Process Standards

Following are some of the product & process standards

Product Standards

Design review form

Requirements document structure

Procedure header format

Project plan format

Change request form

Process Standards

Design review conduct

Submission of documents

Version release process

Change control process

Test recording process

8.4 Quality Planning

Quality Planning should begin at an early stage in the software process. A quality plan should set out the desired product qualities. These include the following:

- a) *Product Introduction*: A description of the product, its intended market and the quality expectations for the product.
- b) *Product Plans*: The critical release dates and responsibilities for the product along with plans for distribution and product servicing.
- c) *Product Descriptions*: The development and service processes which should be used for development & management.
- d) *Quality Goals*: The quality goals and plans for the product, including an identification & justification of critical product quality attributes.
- e) *Risks & Risk Management*: The key is which might affect product quality and the actions to address these risks.

CHAPTER- IX

CONCLUSION AND FUTURE SCOPE

Our Project, encryption & decryption of Files, finished successfully with the main objective getting accomplished. Our system is successfully able to encrypt & decrypt text, audio, video and picture files of known extension efficiently and quickly. The system is user friendly giving least amount of any trouble to its users.

With the completion of the project and with the design of graphical user interface for encryption and decryption of files, we learned not only the technical knowledge for designing and developing such system, but also the managerial 'know how', how to follow a project life cycle to reach the end starting from the scratch.

We have designed a system which encrypts and decrypts any file present on a single system/node where we execute our program; however for some purposes we may need our encryption/decryption system to work on multiple system/nodes as we transfer files from one node to another. Therefore our system can be improved by implementing it on multiple systems and by protecting the files to be accessed by unauthorized users.

- Java Development Kit 1.6.1_10 and JRE 1.5.0_03 should already be installed on the system where we would run our program.
- Programming language: Java 1.6.1_10
- Software Used : Any editor for java such as JCreator 3.10 LE or JEdit can be used for efficient execution of the program.
- Operating System: Windows XP
- Platform Used: Windows

CHAPTER –X

INSTALLATION GUIDE

Contents of the CD and its directory structure

The contents of the CD are as follows:

- Source Codes
- S/W and H/W Requirements
- Compiling and Execution Method
- Test Layout with Expected output

These are discussed as follows.

10.1 S/W and H/W requirements

These are the requirements that are to be fulfilled by the system on which the encryption and decryption of files would take place.

10.1.1 S/W Requirements:

- Java Development Kit 1.6.1_10 and JRE 1.5.0_03 should already be installed on the system where we would run our program.
- Programming Language: Java 1.6.1_10
- Software Used : Any editor for java such as JCreator 3.10 LE or JEdit can be used for efficient execution of the program.
- Operating System: Windows XP
- Platform Used: Windows.

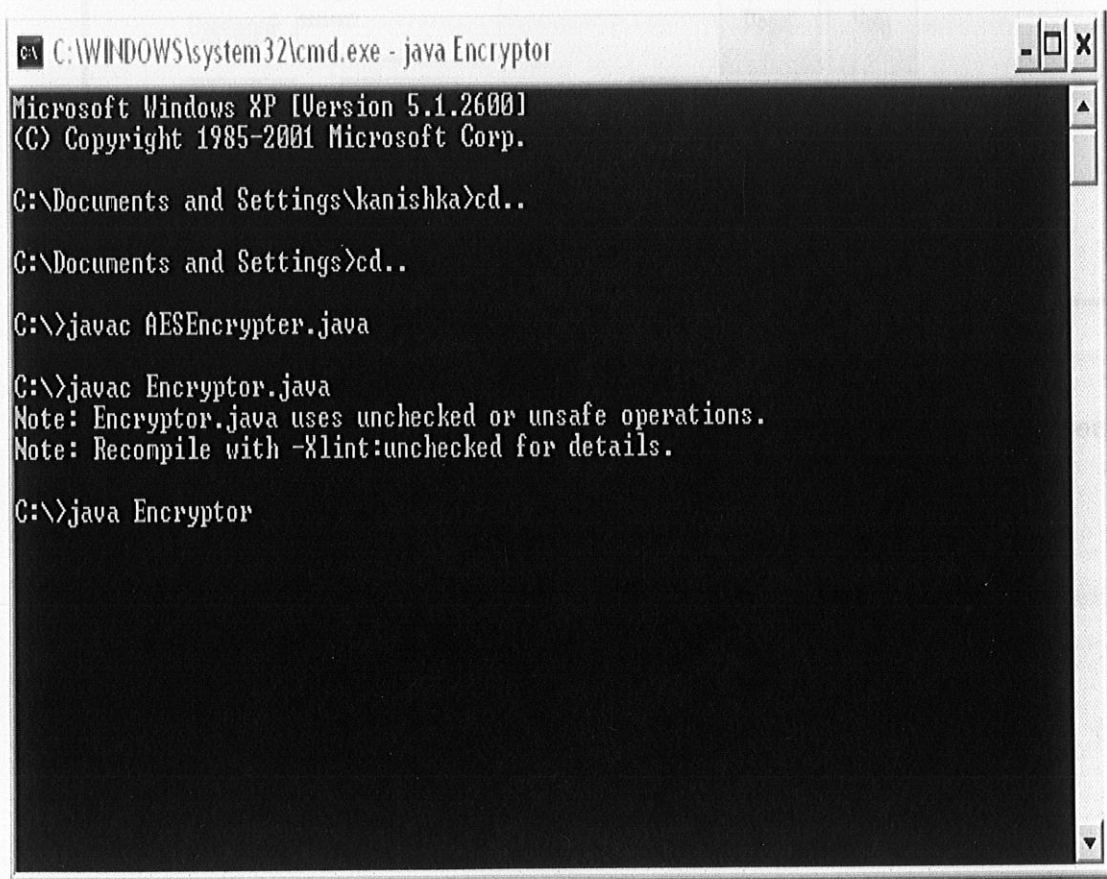
10.1.2 Hardware Requirements:

Performance Requirement:

- Minimum CPU Speed: 166 Mhz
- Minimum HDD: 96 MB
- Minimum RAM: 32 MB

10.2 Installation Procedure

The installation procedure is very simple. We just need to compile all the source codes and run. The user interface which then pops up should be dealt according to the help statements present for each component. The following screenshot will help for installation procedure.



```
C:\WINDOWS\system32\cmd.exe - java Encryptor
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\kanishka>cd..

C:\Documents and Settings>cd..

C:\>javac AESEncrypter.java

C:\>javac Encryptor.java
Note: Encryptor.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

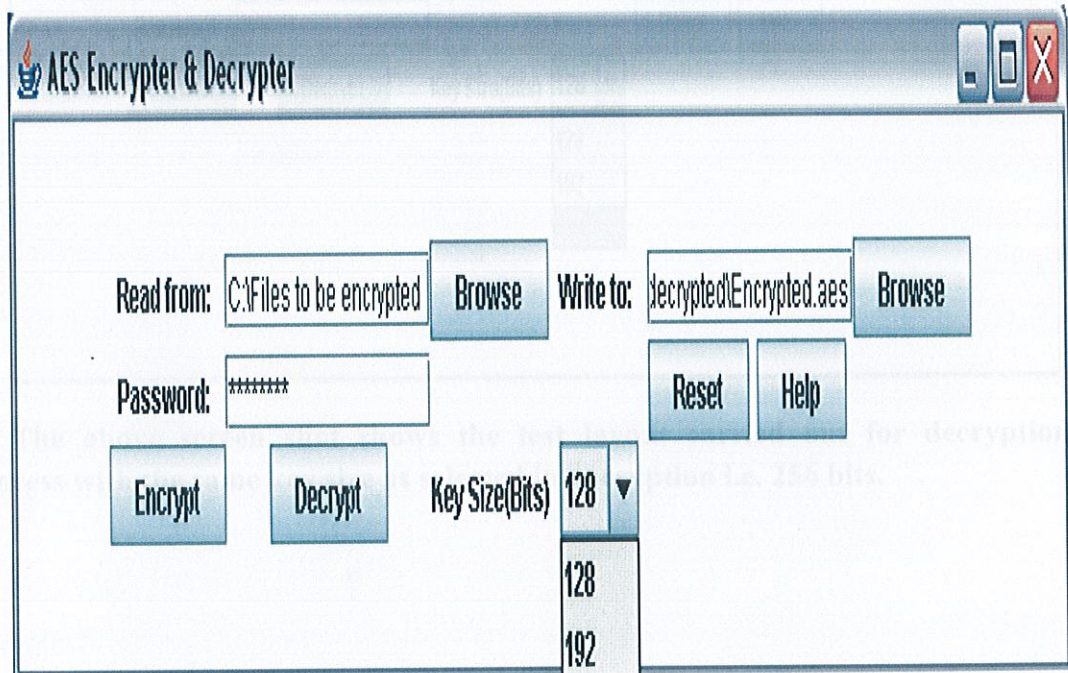
C:\>java Encryptor
```

Here we are just showing the compilation and execution of our main module which is the encryption and decryption of files and its user interface.

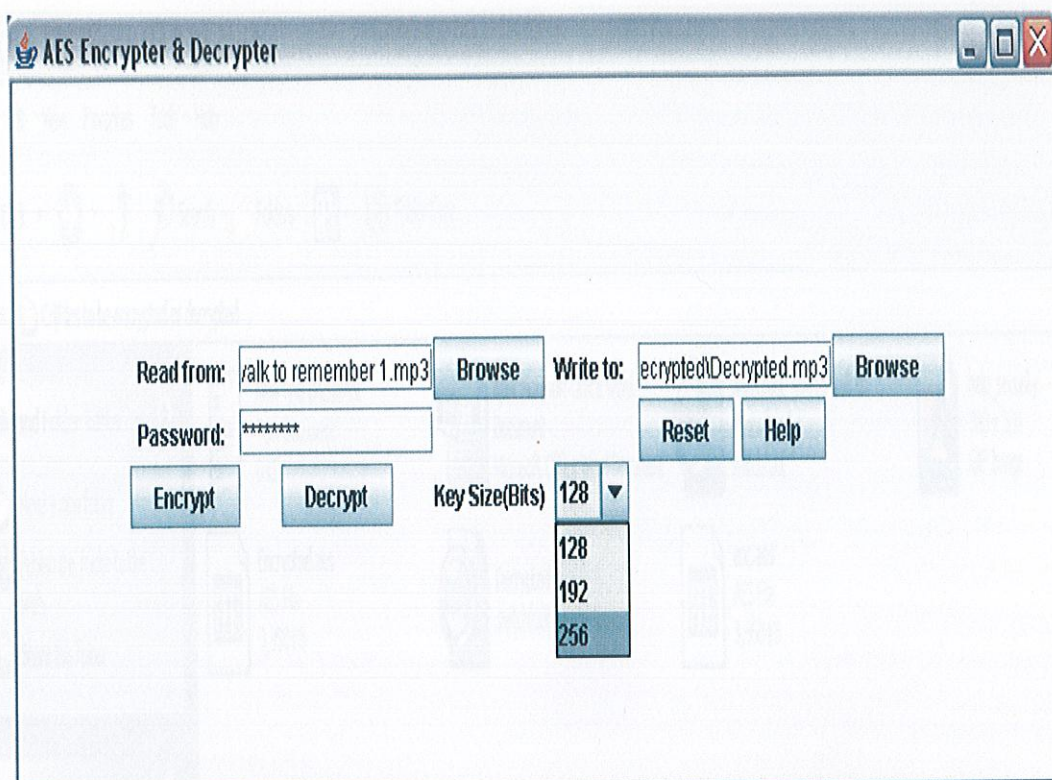
10.3 Compiling and execution method

There are many methods for Compiling and Execution; one is just by using command prompt and using javac statement for compilation and java statement for execution, another method is to execute the file on some standard editor such as JCreator 3.10 LE or JEdit.

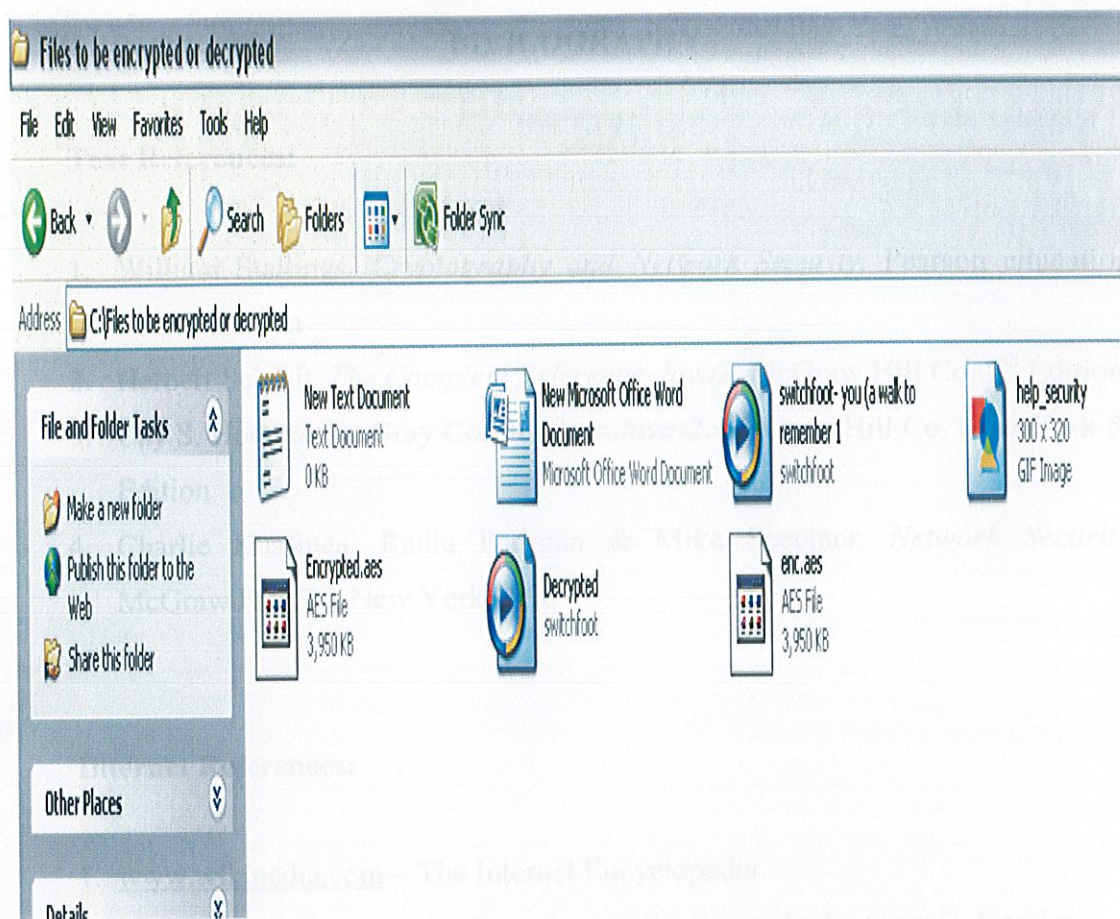
10.4 Test Layout with Expected output.



The above screen shot shows the test layout carried out for encryption process with the key size selected as 256.



The above screen shot shows the test layout carried out for decryption process with the same key size as selected in encryption i.e. 256 bits.



The above screen shot shows the expected output. As the path entered by the user for encrypted file and decrypted file to be saved, the encrypted and decrypted path are saved. The encrypted file has the extension as .aes while decrypted file has the extension as .mp3.

BIBLIOGRAPHY

Text References:

1. William Stallings. *Cryptography and Network Security*. Pearson education. Fourth Edition
2. Herbert Schildt. *The Complete Reference-Java2*. McGraw Hill Co. 7th Edition
3. Cay S. Horstmann, Gray Cornll. *Core Java-2*. McGraw Hill Co. New York 5th Edition
4. Charlie Kuafmaa, Radia Perlman & Mike Speciner. *Network Security*. McGraw Hill Co. New York 1996

Internet References:

1. www.wikipedia.com – The Internet Encyclopedia.
2. www.graykessler.net – An Overview of Cryptography by Gray C. Kessler.
3. www.nist.gov – AES, Question & Answers.
4. www.aeslounge.com – Contains a Comprehensive Bibliography of documents and papers on AES, with access to electronic copies.

Research Papers:

1. Daemen, J., and Rijmen, V. "Rijndael: The Advanced Encryption Standard." Dr,Dobb's Journal, March 2001.
2. Daemen, J., and Rijmen, V. *The Design of Rijndael: The Wide Trail Strategy Explained*. New York, Springer-Verlag. 2002.
3. Landau, S. "Polynomials in the Nation's Service: Using Algebra to Design the Advanced Encryption Standard". American Mathematical Monthly, February 2004.