JSPM's

Jayawantrao Sawant College of Engineering, Hadapsar, Pune-411028

Department of Computer Engineering

Lab Manual

LP II- Information Security

TE Computer (2019 Pattern)

Subject Code: 310254(A)

Academic Year: 2022-2023

Prepared By Prof.A.B.Ghandat

VISSION AND MISSION

Vision and Mission of Institute:

Vision:

To satisfy the aspirations of youth force, who wants to lead nation towards prosperity through techno-economic development.

Mission:

To provide, nurture and maintain an environment of high academic excellence, research and entrepreneurship for all aspiring students, which will prepare them to face global challenges maintaining high ethical and moral standards.

Vision and Mission Department:

Vision:

To be a leading educational center grooming computer engineers to serve the society.

Mission:

M1. To develop computer professionals by providing quality education.

M2. To assimilate academics, research and entrepreneurship skills to accomplish real world challenges.

PROGRAMME OUTCOMES (PO)

PO	Key Points	Description
1	**Engineering knowledge**	Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2	**Problem analysis**	Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences and engineering sciences
3	**Design /development of solutions**	Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety and the cultural, societal and environmental considerations
4	**Conduct investigations of complex problems**	Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data and synthesis of the information to provide valid conclusions
5	**Modern tool usage**	Create, select and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations
6	**The engineer and society**	Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice
7	**Environment and sustainability**	Understand the impact of the professional engineering solutions in societal and environmental contexts and demonstrate the knowledge of and need for sustainable development.
8	**Ethics**	Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice
9	**Individual and team work**	Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10	**Communication**	Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations and give and receive clear instructions
11	**Project management and finance**	Demonstrate knowledge and understanding of the engineering and management Principles and apply these to one's own work, as a member and leader in a team manage projects and in multidisciplinary environments
12	**Life-long learning**	Recognize the need for and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change

SavitribaiPhule Pune University
Third Year of Computer Engineering (2019 Course)
310258: Laboratory Practice II

Home

Teaching Scheme	Credit Scheme	Examination Scheme and Marks
Practical: 04 Hours/Week	**02**	**Term Work: 50 Marks** **Practical: 25 Marks**

Companion Course: Artificial Intelligence (310253), Elective II (310245)

Course Objectives:

- To learn and apply various search strategies for AI
- To Formalize and implement constraints in search problems
- To understand the concepts of Information Security / Augmented and Virtual Reality/Cloud Computing/Software Modeling and Architectures

Course Outcomes:

On completion of the course, learner will be able to

- **Artificial Intelligence**

  CO1: Design system using different informed search / uninformed search or heuristic approaches
  CO2: Apply basic principles of AI in solutions that require problem solving, inference, perception, knowledge representation, and learning
  CO3: Design and develop an expert system

- **Information Security**

  CO4: Use tools and techniques in the area of Information Security
  CO5: Use the knowledge of security for problem solving
  CO6: Apply the concepts of Information Security to design and develop applications

OR

- **Augmented and Virtual Reality**

  CO4: Use tools and techniques in the area of Augmented and Virtual Reality
  CO5: Use the knowledge of Augmented and Virtual Reality for problem solving
  CO6: Apply the concepts of Augmented and Virtual Reality to design and develop applications

OR

- **Cloud Computing**

  CO4: Use tools and techniques in the area of Cloud Computing
  CO5: Use the knowledge of Cloud Computing for problem solving
  CO6: Apply the concepts Cloud Computing to design and develop applications

OR

- **Software Modeling and Architectures**

  CO4: Use tools and techniques in the area Software Modeling and Architectures
  CO5: Use the knowledge of Software Modeling and Architectures for problem solving
  CO6: Apply the concepts Software Modeling and Architectures to design and develop applications

Guidelines for Instructor's Manual

The instructor‘s manual is to be developed as a reference and hands-on resource. It should include prologue (about University/program/ institute/ department/foreword/ preface), curriculum of the course,

conduction and Assessment guidelines, topics under consideration, concept, objectives, outcomes, set of typical applications/assignments/ guidelines, and references.

Guidelines for Student's Laboratory Journal

The laboratory assignments are to be submitted by student in the form of journal. Journal consists of Certificate, table of contents, and handwritten write-up of each assignment (Title, Date of Completion, Objectives, Problem Statement, Software and Hardware requirements, Assessment grade/marks and assessor's sign, Theory- Concept in brief, algorithm, flowchart, test cases, Test Data Set(if applicable), mathematical model (if applicable), conclusion/analysis. Program codes with sample output of all performed assignments are to be submitted as softcopy. As a conscious effort and little contribution towards Green IT and environment awareness, attaching printed papers as part of write-ups and program listing to journal must be avoided. Use of DVD containing students programs maintained by Laboratory In-charge is highly encouraged. For reference one or two journals may be maintained with program prints in the Laboratory.

Guidelines for Laboratory /Term Work Assessment

Continuous assessment of laboratory work should be based on overall performance of Laboratory assignments by a student. Each Laboratory assignment assessment will assign grade/marks based on parameters, such as timely completion, performance, innovation, efficient codes, punctuality and

Guidelines for Practical Examination

Problem statements must be decided jointly by the internal examiner and external examiner. During practical assessment, maximum weightage should be given to satisfactory implementation of the problem statement. Relevant questions may be asked at the time of evaluation to test the student‘s understanding of the fundamentals, effective and efficient implementation. This will encourage, transparent evaluation and fair approach, and hence will not create any uncertainty or doubt in the minds of the students. So, adhering to these principles will consummate our team efforts to the promising start of student's academics.

Guidelines for Laboratory Conduction

The instructor is expected to frame the assignments by understanding the prerequisites, technological aspects, utility and recent trends related to the topic. The assignment framing policy need to address the average students and inclusive of an element to attract and promote the intelligent students. Use of open source software is encouraged. Based on the concepts learned. Instructor may also set one assignment or mini-project that is suitable to respective branch beyond the scope of syllabus.

Operating System recommended :- 64-bit Windows OS and Linux

Programming tools recommended: -

Information Security : - C/C++/Java

Augmented and Virtual Reality :- Unity, C#, Blender, VRTK, ARTK, Vuforia

VR Devices: HTC Vive, Google Daydream and Samsung gear VR.

Cloud Computing :-

Software Modeling and Architectures: Front end:HTML5, Bootstrap, jQuery, JS etc.

Backend: MySQL/MongoDB/NodeJS

Virtual Laboratory:

Software Modeling and Architectures : http://vlabs.iitkgp.ernet.in/se

Information Security : http://cse29-iiith.vlabs.ac.in

Part I : Artificial Intelligence

Suggested List of Laboratory Experiments/Assignments

Sr. No.	Group A All assignments are compulsory
1.	Implement depth first search algorithm and Breadth First Search algorithm, Use an undirected graph and develop a recursive algorithm for searching all the vertices of a graph or tree data structure.
2.	Implement A star Algorithm for any game search problem.

3.	Implement Greedy search algorithm for any of the following application: I. Selection Sort II. Minimum Spanning Tree III. Single-Source Shortest Path Problem IV. Job Scheduling Problem V. Prim's Minimal Spanning Tree Algorithm VI. Kruskal's Minimal Spanning Tree Algorithm VII. Dijkstra's Minimal Spanning Tree Algorithm
	Group B
4.	Implement a solution for a Constraint Satisfaction Problem using Branch and Bound and Backtracking for n-queens problem or a graph coloring problem.
5.	Develop an elementary chatbot for any suitable customer interaction application.
	Group C
6.	Implement any one of the following Expert System I. Information management II. Hospitals and medical facilities III. Help desks management IV. Employee performance evaluation V. Stock market trading VI. Airline scheduling and cargo schedules
Part II : Elective II	
Suggested List of Laboratory Experiments/Assignments	
Sr. No.	**Assignment Name**
	Information Security (Any five)
1.	Write a Java/C/C++/Python program that contains a string (char pointer) with a value \Hello World'. The program should AND or and XOR each character in this string with 127 and display the result.
2.	Write a Java/C/C++/Python program to perform encryption and decryption using the method of Transposition technique.
3.	Write a Java/C/C++/Python program to implement DES algorithm.
4.	Write a Java/C/C++/Python program to implement AES Algorithm.
5.	Write a Java/C/C++/Python program to implement RSA algorithm.
6.	Implement the Diffie-Hellman Key Exchange mechanism using HTML and JavaScript. Consider the end user as one of the parties (Alice) and the JavaScript application as other party (bob).
7.	Calculate the message digest of a text using the MD5 algorithm in JAVA.
	Cloud Computing (All assignments are compulsory)
1.	Case study on Microsoft azure to learn about Microsoft Azure is a cloud computing platform and infrastructure, created by Microsoft, for building, deploying and managing applications and services through a global network of Microsoft-managed data centers. OR Case study on Amazon EC2 and learn about Amazon EC2 web services.
2.	Installation and configure Google App Engine. OR

	Installation and Configuration of virtualization using KVM.
3.	Creating an Application in SalesForce.com using Apex programming Language.
4.	Design and develop custom Application (Mini Project) using Salesforce Cloud.
5.	**Mini-Project** Setup your own cloud for Software as a Service (SaaS) over the existing LAN in your laboratory. In this assignment you have to write your own code for cloud controller using open-source technologies to implement **with HDFS**. Implement the basic operations may be like to divide the file in segments/blocks and upload/ download file on/from cloud in encrypted form.
	Augmented and Virtual Reality **(Assignments 1,2, 3,7 are mandatory, any 2 from 4, 5 & 6)**
1.	Installation of Unity and Visual Studio, setting up Unity for VR development, understanding documentation of the same.
2.	Demonstration of the working of HTC Vive, Google Daydream or Samsung gear VR.
3.	Develop a scene in Unity that includes: i. A cube, plane and sphere, apply transformations on the 3 game objects. ii. Add a video and audio source.
4.	Develop a scene in Unity that includes a cube, plane and sphere. Create a new material and texture separately for three Game objects. Change the color, material and texture of each Game object separately in the scene. Write a C# program in visual studio to change the color and material/texture of the game objects dynamically on button click.
5.	Develop and deploy a simple marker based AR app in which you have to write a C# program to play video on tracking a particular marker.
6.	Develop and deploy an AR app, implement the following using Vuforia Engine developer portal: i. Plane detection ii. Marker based Tracking(Create a database of objects to be tracked in Vuforia) iii. Object Tracking
7.	**Mini-Projects/ Case Study** Create a multiplayer VR game (battlefield game). The game should keep track of score, no. of chances/lives, levels (created using different scenes), involve interaction, animation and immersive environment. **OR** Create a treasure hunt AR application which should have the following features: i. A help button for instruction box to appear. ii. A series of markers which would give hints on being scanned. iii. Involve interaction, sound, and good UI.
	Software Modeling and Architectures **(Problem statement 1, 2 , 3 or 4, Problem statement 5 and 6 are mandatory)**
1.	Consider a library, where a member can perform two operations: issue book and return it. A book is issued to a member only after verifying his credentials. Develop a use case diagram for the given library system by identifying the actors and use cases and associate the use cases with the actors by drawing a use case diagram. Use UML tool.
2.	Consider online shopping system. Perform the following tasks and draw the class diagram using UML tool. Represent the individual classes, and objects Add methods Represent relationships and other classifiers like interfaces
3.	Consider the online shopping system in the assignment 2.

	Draw the sequence diagram using UML tool to show message exchanges
4.	Consider your neighboring travel agent from whom you can purchase flight tickets. To book a ticket you need to provide details about your journey i.e., on which date and at what time you would like to travel. You also need to provide your address. The agency has recently been modernized. So, you can pay either by cash or by card. You can also cancel a booked ticket later if you decide to change your plan. In that case you need to book a new ticket again. Your agent also allows you to book a hotel along with flight ticket. While cancelling a flight ticket you can also cancel hotel booking. Appropriate refund as per policy is made in case of cancellation. Perform the following tasks and draw the use case diagram using UML tool. a. Identify the use cases from a given non-trivial problem statement. b. Identify the primary and secondary actors for a system. **c.** Use to generalization of use cases and «include» stereotypes to prevent redundancy in the coding phase
Mini-Projects	
5.	Select a moderately complex system and narrate concise requirement Specification for the same. Design the system indicating system elements organizations using applicable architectural styles and design patterns with the help of a detailed Class diagram depicting logical architecture. Specify and document the architecture and design pattern with the help of templates. Implement the system features and judge the benefits of the design patterns accommodated.

Learning Resources

Text Books:

Artificial Intelligence

1. Stuart Russell and Peter Norvig, “Artificial Intelligence: A Modern Approach”, Third edition, Pearson, 2003, ISBN :10: 0136042597
2. Deepak Khemani, “A First Course in Artificial Intelligence”, McGraw Hill Education(India), 2013, ISBN : 978-1-25-902998-1
3. Elaine Rich, Kevin Knight and Nair, “Artificial Intelligence”, TMH, ISBN-978-0-07-008770-5

Information Security

1. Atul Kahate, “Cryptography and Network Security”, 3e, McGraw Hill Education
2. Prakash C. Gupta, “Cryptography and Network Security”, PHI
3. V.K. Pachghare, “Cryptography and Information Security”, PHI Learning

Cloud Computing

1. A. Srinivasan, J. Suresh,” Cloud Computing: A Practical Approach for Learning and Implementation”, Pearson, ISBN: 978-81-317-7651-3
2. Rajkumar Buyya, Christian Vecchiola, S. Thamarai Selvi, “Mastering Cloud Computing”, McGraw Hill Education, ISBN-13:978-1-25-902995-0

Augmented and Virtual Reality

1. William R Sherman and Alan B Craig, “Understanding Virtual Reality: Interface, Application and Design”, (The Morgan Kaufmann Series in Computer Graphics)”. Morgan Kaufmann Publishers, San Francisco, CA, 2002
2. Alan B Craig, “Understanding Augmented Reality, Concepts and Applications”, Morgan Kaufmann Publishers, ISBN:978-0240824086

Software Modelling and Architectures

1. Jim Arlow, Ila Neustadt, “UML 2 and the unified process –practical object-oriented analysis and design”, Addison Wesley, Second edition, ISBN 978-0201770605
2. Len Bass, Paul Clements, Rick Kazman, "Software Architecture in Practice", Second Edition, Pearson ,ISBN 978-81-775-8996-2
3. Hassan Gomaa, “Software Modeling and Design- UML, Use cases, Patterns and Software Architectures”, Cambridge University Press, 2011, ISBN 978-0-521-76414-8
4. Erich Gamma, “Design Patterns”, Pearson, ISBN 0-201-63361-2

Reference Books:

1. Nilsson Nils J , “Artificial Intelligence: A new Synthesis”, Morgan Kaufmann Publishers Inc. San Francisco, CA, ISBN: 978-1-55-860467-4
2. Patrick Henry Winston, “Artificial Intelligence”, Addison-Wesley Publishing Company, ISBN: 0-201-53377-4
3. Andries P. Engelbrecht, “Computational Intelligence: An Introduction”, 2nd Edition-Wiley India-ISBN: 978-0-470-51250-0

Information Security

1. William Stallings, Lawrie Brown, “Computer Security Principles and Practice”, 3rd_Edition, Pearson
2. William Stallings, “Cryptography and Network Security Principals and Practice”, Fifth edition, Pearson
3. Nina Godbole, Sunit Belapure, “Cyber Security”, Wiley, ISBN: 978-81-265-2179-1

Augmented and Virtual Reality

1. Steven M. LaValle, “Virtual Reality”, Cambridge University Press, 2016
2. Alan B Craig, William R Sherman and Jeffrey D Will, “Developing Virtual Reality Applications: Foundations of Effective Design”, Morgan Kaufmann, 2009.
3. Schmalstieg / Hollerer, “Augmented Reality: Principles & Practice”, Pearson Education India; First edition (12 October 2016),ISBN-10: 9332578494
4. Sanni Siltanen, “Theory and applications of marker-based augmented reality”, Julkaisija – Utgivare Publisher. 2012. ISBN 978-951-38-7449-0

Cloud Computing

1. James Bond ,“The Enterprise Cloud”, O'Reilly Media, Inc. ISBN: 9781491907627
2. Dr. Kris Jamsa, “Cloud Computing: SaaS, PaaS, IaaS, Virtualization and more”, Wiley Publications, ISBN: 978-0-470-97389-9
3. Anthony T. Velte Toby J. Velte, Robert Elsenpeter, “Cloud Computing: A Practical Approach”, 2010, The McGraw-Hill.

Software Modelling and Architectures

1. Gardy Booch, James Rambaugh, Ivar Jacobson, “The unified modeling language user guide” , Pearson Education, Second edition, 2008, ISBN 0-321-24562-8.
2. Lan Sommerville, “Software Engineering”, 9th edition, ISBN-13: 978-0-13-703515-1 ISBN-10: 0-13-703515-2.

@The CO-PO Mapping Matrix

CO/PO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	2	-	2	-	3	-	-	2	2	2	1	2

CO2	1	-	2	2	3	2	-	2	2	2	1	2
CO3	1	-	2	2	3	2	-	2	2	2	2	2
CO4	1	-	2	-	3	-	-	2	2	2	2	2
CO5	1	-	2	-	3	-	-	2	2	2	2	2
CO6	1	-	2	-	3	-	-	2	2	2	2	2

EXPERIMENT NO 1

AIM:
Write a C program that contains a string (char pointer) with a value \Hello World'. The program should AND , OR and XOR each character in this string with 127 and display the result

THEORY :
It is expected to see "garbage" with OR or XOR.

Your string Hello World is pure ASCII, so it consists of characters with encoding values less than 127.

A bitwise AND with 127 does not change the value, so you will see Hello World.

A bitwise OR with 127 will set all values to 127 in your case. For characters from other languages you may get values above 127.

A bitwise XOR with 127 will invert the 7 low bits of every character resulting in other characters which may be printable characters or not.

That means when you print the characters you will see "garbage".

Sample Code :

```
void main(){
   char str[]="Hello World";
   int i,len;
   len = strlen(str);
   for(i=0;i<len;i++){
         printf("%c",str[i]&127);
   }
   printf("\n");
   for(int i=0;i<len;i++){
         printf("%c",str[i]^127);
   }
   printf("\n");
   for(int i=0;i<len;i++){
         printf("%c",str[i]|127);
   }
   printf("\n");
}
```

EXPERIMENT 2

AIM :

Write a Java/C/C++/Python program to perform encryption and decryption using the method of Transposition technique.

THEORY :

Encrypting with the Transposition Cipher

Instead of replacing characters with other characters, the transposition cipher jumbles up the message's symbols into an order that makes the original message unreadable. Before we start writing code, let's encrypt the message "Common sense is not so common." with pencil and paper. Including the spaces and punctuation, this message has 30 characters. We will use the number 8 for the key.

The first step is to draw out a number of boxes equal to the key. We will draw 8 boxes since our key for this example is 8:

The second step is to start writing the message you want to encrypt into the boxes, with one character for each box. Remember that spaces are a character (this book marks the boxes with (s) to indicate a space so it doesn't look like an empty box).

C	o	m	m	o	n	(space)	s

We only have 8 boxes but there are 30 characters in the message. When you run out of boxes, draw another row of 8 boxes under the first row. Keep creating new rows until you have written out the full message:

1st	2nd	3rd	4th	5th	6th	7th	8th
C	o	m	m	o	n	(space)	s
e	n	s	e	(space)	i	s	(space)
n	o	t	(space)	s	o	(space)	c
o	m	m	o	n	.		

We shade in the two boxes in the last row to remind us to ignore them. The ciphertext is the letters read from the top left box going down the column. “C”, “e”, “n”, and “o” are from the 1st column. When you get to the last row of a column, move to the top row of the next column to the right. The next characters are “o”, “n”, “o”, “m”. Ignore the shaded boxes.

The ciphertext is “Cenoonommstmme oo snnio. s s c”, which is sufficiently scrambled to keep someone from figuring out the original message by looking at it.

The steps for encrypting are:

1. Count the number of characters in the message and the key.
2. Draw a number of boxes equal to the key in a single row. (For example, 12 boxes for a key of 12.)
3. Start filling in the boxes from left to right, with one character per box.
4. When you run out of boxes and still have characters left, add another row of boxes.
5. Shade in the unused boxes in the last row.
6. Starting from the top left and going down, write out the characters. When you get to the bottom of the column, move to the next column to the right. Skip any shaded boxes. This will be the ciphertext.

Source Code of the Transposition Cipher Encryption Program

Open a new file editor window by clicking on **File ► New Window**.

Type in the following code into the file editor, and then save it as *transpositionEncrypt.py*.

Press **F5** to run the program. Note that first you will need to download the *pyperclip.py* module and place this file in the same directory as the *transpositionEncrypt.py* file.

```
1. # Transposition Cipher Encryption
 2. # http://inventwithpython.com/hacking (BSD Licensed)
 3.
 4. import pyperclip
 5.
 6. def main():
 7.     myMessage = 'Common sense is not so common.'
 8.     myKey = 8
 9.
10.     ciphertext = encryptMessage(myKey, myMessage)
11.
12.     # Print the encrypted string in ciphertext to the screen, with
13.     # a | (called "pipe" character) after it in case there are spaces at
14.     # the end of the encrypted message.
```

```
    print(ciphertext + '|')

    # Copy the encrypted string in ciphertext to the clipboard.
    pyperclip.copy(ciphertext)


def encryptMessage(key, message):
    # Each string in ciphertext represents a column in the grid.
    ciphertext = [''] * key

    # Loop through each column in ciphertext.
    for col in range(key):
        pointer = col

        # Keep looping until pointer goes past the length of the message.
        while pointer < len(message):
            # Place the character at pointer in message at the end of the
            # current column in the ciphertext list.
            ciphertext[col] += message[pointer]

            # move pointer over
            pointer += key

    # Convert the ciphertext list into a single string value and return it.
    return ''.join(ciphertext)


# If transpositionEncrypt.py is run (instead of imported as a module) call
# the main() function.
if __name__ == '__main__':
    main()
```

```
# Transposition Cipher Encryption
 # http://inventwithpython.com/hacking (BSD Licensed)
import pyperclip
def main():
    myKey = 8
    myMessage = 'Common sense is not so common.'
    ciphertext = encryptMessage(myKey, myMessage)
   # Print the encrypted string in ciphertext to the screen, with
```

```
    # a | (called "pipe" character) after it in case there are spaces at
    # the end of the encrypted message.
     print(ciphertext + '|')
    # Copy the encrypted string in ciphertext to the clipboard.
def encryptMessage(key, message):
     # Each string in ciphertext represents a column in the grid.
    ciphertext = [''] * key
    for col in range(key):
        pointer = col
        # Keep looping until pointer goes past the length of the message.
        while pointer < len(message):
            # Place the character at pointer in message at the end of the
            # current column in the ciphertext list.
            ciphertext[col] += message[pointer]
            # move pointer over
            pointer += key

    # Convert the ciphertext list into a single string value and return it.
    return ''.join(ciphertext)
# If transpositionEncrypt.py is run (instead of imported as a module) call
# the main() function.
if __name__ == '__main__':
    main()
```

Sample Run of the Transposition Cipher Encryption Program

When you run the above program, it produces this output:

```
Cenoonommstmme oo snnio. s s c|
```

This ciphertext (without the pipe character at the end) is also copied to the clipboard, so you can paste it into an email to someone. If you want to encrypt a different message or use a different key, change the value assigned to the `myMessage` and `myKey` variables on lines 7 and 8. Then run the program again.

How the Program Works

transpositionEncrypt.py

```
1. # Transposition Cipher Encryption
2. # http://inventwithpython.com/hacking (BSD Licensed)
3.
```

```
4. import pyperclip
```

The transposition cipher program will copy the encrypted text to the clipboard

So first we will import the `pyperclip` module so it can call `pyperclip.copy()`.

Creating Your Own Functions with `def` Statements

transpositionEncrypt.py

```
6. def main():
7.     myMessage = 'Common sense is not so common.'
8.     myKey = 8
```

You can create your own functions with a `def` statement like the one on line 6.

The `def` statement on line 6 isn't a call to a function named `main()`.

Instead, the `def` statement means we are creating, or **defining**, a new function named `main()` that we can call later in our program. When the execution reaches the `def` statement Python will *define* this function.

We can then call it the same way we call other functions. When we *call* this function, the execution moves inside of the block of code following the `def` statement.

On lines 7 and 8, the variables `myMessage` and `myKey` will store the plaintext message to encrypt and the key used to do the encryption.

```
10.     ciphertext = encryptMessage(myKey, myMessage)
```

The code that does the actual encrypting will be put into a function we define on line 21 named `encryptMessage()`.

This function will take two arguments: an integer value for the key and a string value for the message to encrypt. When passing multiple arguments to a function call, separate the arguments with a comma.

The return value of `encryptMessage()` will be a string value of the encrypted ciphertext. (The code in this function is explained next.) This string will be stored in a variable named `ciphertext`.

```
12.     # Print the encrypted string in ciphertext to the screen, with
13.     # a | (called "pipe" character) after it in case there are spaces at
14.     # the end of the encrypted message.
15.     print(ciphertext + '|')
16.
17.     # Copy the encrypted string in ciphertext to the clipboard.
18.     pyperclip.copy(ciphertext)
```

The ciphertext message is printed to the screen on line 15 and copied to the clipboard on line 18.

The program prints a | character (called the "pipe" character) at the end of the message so that the user can see any empty space characters at the end of the ciphertext.

Line 18 is the last line of the `main()` function. After it executes, the program execution will return to the line after the line that called it.

The call to `main()` is on line 45 and is the last line in the program, so after execution returns from `main()` the program will exit.

Parameters

```
21. def encryptMessage(key, message):
```

The code in the `encryptMessage()` function does the actual encryption. The `key` and `message` text in between the parentheses next to `encryptMessage()`'s `def` statement shows that the `encryptMessage()` function takes two parameters.

Parameters are the variables that contain the arguments passed when a function is called. Parameters are automatically deleted when the function returns. (This is just like how variables are forgotten when a program exits.)

When the `encryptMessage()` function gets called from line 10, two argument values are passed (on line 10, they are the values in `myKey` and `myMessage`). These values get assigned to the parameters `key` and `message` (which you can see on line 21) when the execution moves to the top of the function.

A parameter is a variable name in between the parentheses in the `def` statement. An argument is a value that is passed in between the parentheses for a function call.

Python will raise an error message if you try to call a function with too many or too few arguments for the number of parameters the function has.

The List Data Type

```
22.     # Each string in ciphertext represents a column in the grid.
23.     ciphertext = [''] * key
```

Line 23 uses a new data type called the **list** data type. A list value can contain other values. Just like how strings begin and end with quotes, a list value begins with a `[` open bracket and ends with `]` close bracket. The values stored inside the list are typed within the brackets. If there is more than one value in the list, the values are separated by commas.

The Transposition Encryption Algorithm

We need to translate these paper-and-pencil steps into Python code. Let's take a look at encrypting the string `'Common sense is not so common.'` with the key `8`. If we wrote out the boxes with pencil and paper, it would look like this:

C	o	m	m	o	n	(space)	s
e	n	s	e	(space)	i	s	(space)
n	o	t	(space)	s	o	(space)	c
o	m	m	o	n	.		

Add the index of each letter in the string to the boxes. (Remember, indexes begin with `0`, not `1`.)

C 0	o 1	m 2	m 3	o 4	n 5	(space) 6	s 7
e 8	n 9	s 10	e 11	(space) 12	i 13	s 14	(space) 15
n 16	o 17	t 18	(space) 19	s 20	o 21	(space) 22	c 23
o 24	m 25	m 26	o 27	n 28	. 29		

We can see from these boxes that the first column has the characters at indexes `0`, `8`, `16`, and `24` (which are `'C'`, `'e'`, `'n'`, and `'o'`). The next column has the characters at indexes `1`, `9`, `17`, and `25` (which are `'o'`, `'n'`, `'o'` and `'m'`). We can see a pattern emerging: The n^{th} column will have all the characters in the string at indexes 0 + n, 8 + n, 16 + n, and 24 + n:

C 0+0=0	o 1+0=1	m 2+0=2	m 3+0=3	o 4+0=4	n 5+0=5	(s) 6+0=6	s 7+0=7
e 0+8=8	n 1+8=9	s 2+8=10	e 3+8=11	(s) 4+8=12	i 5+8=13	s 6+8=14	(s) 7+8=15
n 0+16=16	o 1+16=17	t 2+16=18	(s) 3+16=19	s 4+16=20	o 5+16=21	(s) 6+16=22	c 7+16=23
o 0+24=24	m 1+24=25	m 2+24=26	o 3+24=27	n 4+24=28	. 5+24=29		

There is an exception for the 6th and 7th columns, since 24 + 6 and 24 + 7 are greater than 29, which is the largest index in our string. In those cases, we only use 0, 8, and 16 to add to n (and skip 24).

What's so special about the numbers 0, 8, 16, and 24? These are the numbers we get when, starting from 0, we add the key (which in this example is 8). 0 + 8 is 8, 8 + 8 is 16, 16 + 8 is 24. 24 + 8 would be 32, but since 32 is larger than the length of the message, we stop at 24.

So, for the nth column's string we start at index n, and then keep adding 8 (which is the key) to get the next index. We keep adding 8 as long as the index is less than 30 (the message length), at which point we move to the next column.

If we imagine a list of 8 strings where each string is made up of the characters in each column, then the list value would look like this:

```
['Ceno', 'onom', 'mstm', 'me o', 'o sn', 'nio.', ' s ', 's c']
```

This is how we can simulate the boxes in Python code. First, we will make a list of blank strings. This list will have a number of blank strings equal to the key because each string will represent a column of our paper-and-pencil boxes. (Our list will have 8 blank strings since we are using the key 8 in our example.) Let's look at the code.

```
22.     # Each string in ciphertext represents a column in the grid.
23.     ciphertext = [''] * key
```

The `ciphertext` variable will be a list of string values. Each string in the `ciphertext` variable will represent a column of the grid. So `ciphertext[0]` is the leftmost column, `ciphertext[1]` is the column to the right of that, and so on.

The string values will have all the characters that go into one column of the grid. Let's look again at the grid from the "Common sense is not so common." example earlier in this chapter (with column numbers added to the top):

0	1	2	3	4	5	6	7
C	o	m	m	o	n	(space)	s
e	n	s	e	(space)	i	s	(space)
n	o	t	(space)	s	o	(space)	c
o	m	m	o	n	.		

The `ciphertext` variable for this grid would look like this:

```
>>> ciphertext = ['Ceno', 'onom', 'mstm', 'me o', 'o sn', 'nio.', ' s ', 's c']
>>> ciphertext[0]
'Ceno'
```

The first step to making this list is to create as many blank strings in the `ciphertext` list as there are columns.

Since the number of columns is equal to the key, we can use list replication to multiply a list with one blank string value in it by the value in `key`.

This is how line 23 evaluates to a list with the correct number of blank strings.

```
25.     # Loop through each column in ciphertext.
26.     for col in range(key):
27.         pointer = col
```

The next step is to add text to each string in `ciphertext`. The `for` loop on line 26 will iterate once for each column, and the `col` variable will have the correct integer value to use for the index to `ciphertext`. The `col` variable will be set to `0` for the first iteration through the `for` loop, then `1` on the second iteration, then `2` and so on. This way the expression `ciphertext[col]` will be the string for the `col`th column of the grid.

Meanwhile, the `pointer` variable will be used as the index for the string value in the `message` variable. On each iteration through the loop, `pointer` will start at the same value as `col` (which is what line 27 does.)

transpositionEncrypt.py

```
29.         # Keep looping until pointer goes past the length of the message.
30.         while pointer < len(message):
31.             # Place the character at pointer in message at the end of the
32.             # current column in the ciphertext list.
33.             ciphertext[col] += message[pointer]
34.
35.             # move pointer over
36.             pointer += key
```

Inside the `for` loop that started on line 26 is a `while` loop that starts on line 30. For each column, we want to loop through the original `message` variable and pick out every `key`th character. (In the example we've been using, we want every 8th character since we are using a key of `8`.) On line 27 for the first iteration of the `for` loop, `pointer` was set to `0`.

While the value in `pointer` is less than the length of the `message` string, we want to add the character at `message[pointer]` to the end of the `col`th string in `ciphertext`. We add `8` (that is, the value in `key`) to `pointer` each time through the loop on line 36. The first time it is `message[0]`, the second time `message[8]`, the third time `message[16]`, and the fourth time `message[24]`. Each of these single character strings are concatenated to the end of `ciphertext[col]` (and since `col` is `0` on the first time through the loop, this is `ciphertext[0]`).

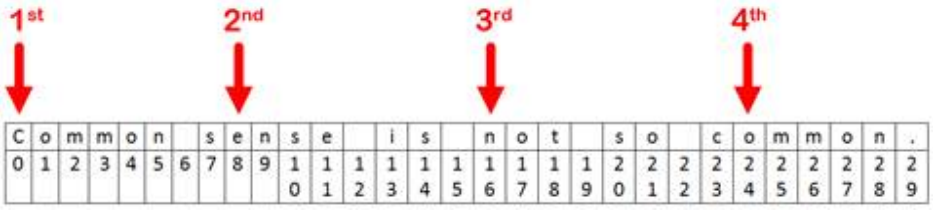


Figure 8-2. Arrows pointing to what `message[pointer]` refers to during the first iteration of the for loop when col is set to `0`.

Figure 8-2 shows the characters at these indexes, they will be concatenated together to form the string `'Ceno'`. Remember that we want the value in ciphertext to eventually look like this:

```
>>> ciphertext = ['Ceno', 'onom', 'mstm', 'me o', 'o sn', 'nio.', ' s ', 's c']
>>> ciphertext[0]
'Ceno'
>>>
```

Storing `'Ceno'` as the first string in the `ciphertext` list is our first step.

On the next iteration of the `for` loop, `col` will be set to `1` (instead of `0`) and `pointer` will start at the same value as `col`. Now when we add `8` to `pointer` on each iteration of line 30's `while` loop, the indexes will be `1`, `9`, `17`, and `25`.

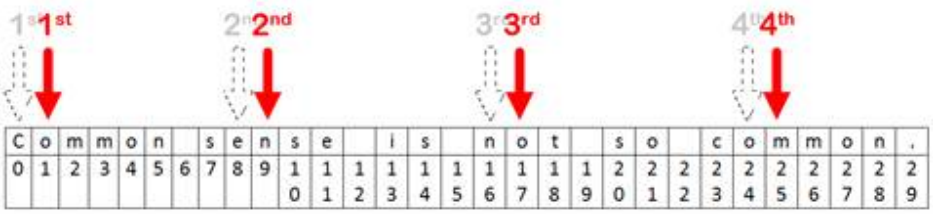


Figure 8-3. Arrows pointing to to what `message[pointer]` refers to during the second iteration of the for loop when col is set to `1`.

As `message[1]`, `message[9]`, `message[17]`, and `message[25]` are concatenated to the end of `ciphertext[1]`, they form the string `'onom'`. This is the second column of our grid.

Once the `for` loop has finished looping for the rest of the columns, the value in `ciphertext` will be `['Ceno', 'onom', 'mstm', 'me o', 'o sn', 'nio.', ' s ', 's c']`. We will use the `join()` string method to convert this list of strings into a single string.

eturn Values and `return` Statements

transpositionEncrypt.py

```
38.     # Convert the ciphertext list into a single string value and return it.
39.     return ''.join(ciphertext)
```

Our use of the `join()` method isn't nearly as complicated as the previous example. We just want to call `join()` on the blank string and pass `ciphertext` as the argument so that the strings in the `ciphertext` list are joined together (with nothing in between them).

Remember that a function (or method) call always evaluates to a value. We say that this is the value *returned* by the function or method call, or that it is the *return value* of the function. When we create our own functions with a `def` statement, we use a `return` statement to tell what the return value for our function is.

A `return` statement is the `return` keyword followed by the value to be returned. We can also use an expression instead of a value. In that case the return value will be whatever value that expression evaluates to.

```
38.     # Convert the ciphertext list into a single string value and return it.
39.     return ''.join(ciphertext)
```

The `encryptMessage()` function's `return` statement returns a string value that is created by joining all of the strings in the `ciphertext` list. This final string is the result of our encryption code.

transpositionEncrypt.py

```
42. # If transpositionEncrypt.py is run (instead of imported as a module) call
43. # the main() function.
44. if __name__ == '__main__':
45.     main()
```

We can turn our transposition encryption program into a module with a special trick involving the `main()` function and a variable named `__name__`.

When a Python program is run, there is a special variable with the name `__name__` (that's two underscores before "name" and two underscores after) that is assigned the string value `'__main__'` (again, two underscores before and after "main") even before the first line of your program is run.

At the end of our script file (and, more importantly, after all of our `def` statements), we want to have some code that checks if the `__name__` variable has the `'__main__'` string assigned to it. If so, we want to call the `main()` function.

This `if` statement on line 44 ends up actually being one of the first lines of code executed when we press **F5** to run our transposition cipher encryption program (after the `import` statement on line 4 and the `def` statements on lines 6 and 21).

The reason we set up our code this way is although Python sets `__name__` to `'__main__'` when the program is run, it sets it to the string `'transpositionEncrypt'` if our program is imported by a

different Python program. This is how our program can know if it is being run as a program or imported by a different program as a module.

Just like how our program imports the `pyperclip` module to call the functions in it, other programs might want to import *transpositionEncrypt.py* to call its `encryptMessage()` function. When an `import` statement is executed, Python will look for a file for the module by adding ".py" to the end of the name. (This is why `import pyperclip` will import the *pyperclip.py* file.)

When a Python program is imported, the `__name__` variable is set to the filename part before ".py" and then runs the program. When our *transpositionEncrypt.py* program is imported, we want all the `def` statements to be run (to define the `encryptMessage()` function that the importing program wants to use), but we don't want it to call the `main()` function because that will execute the encryption code for `'Common sense is not so common.'` with key `8`.

That is why we put that part of the code inside a function (which by convention is named `main()`) and then add code at the end of the program to call `main()`. If we do this, **then our program can both be run as a program on its own and also imported as a module by another program.**

Key Size and Message Length

Notice what happens when the message length is less than twice the key size:

C	o	m	m	o	n	(s)	s	e	n	s	e	(s)	i	s	(s)	n	o	t	(s)	s	o	(s)	c	o
m	m	o	n	.																				

When using a key of 25, the "Common sense is not so common." message encrypts to "Cmommomno.n sense is not so co". Part of the message isn't encrypted! This happens whenever key size becomes more than twice the message length, because that causes there to only be one character per column and no characters get scrambled for that part of the message.

Because of this, the transposition cipher's key is limited to half the length of the message it is used to encrypt. The longer a message is, the more possible keys that can be used to encrypt it.

ASSIGNMENT NO. 04

AIM: To implement a Simplified Advanced Encryption Standard (S-AES) algorithm.

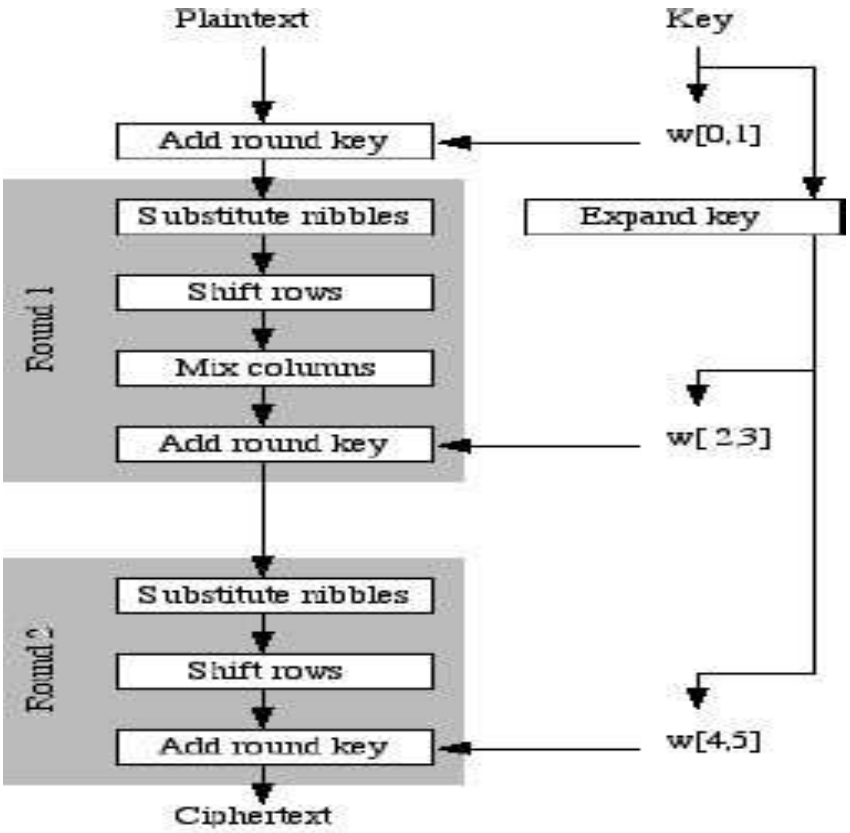
OBJECTIVES:

- The Basic Concepts of A S-AES
- General structure S-AES.
- Logical implementation of S-AES.

THEORY

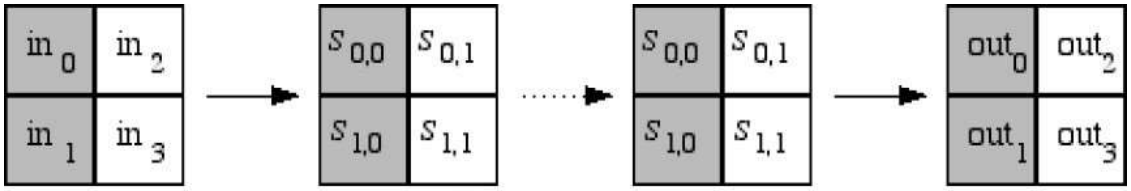
4. INTRODUCTION

S-AES is to AES as S-DES is to DES. In fact, the structure of S-AES is exactly the same as AES. The differences are in the key size (16 bits), the block size (16 bits) and the number of rounds (2 rounds). Here is an overview:



S-AES Encryption Overview

Substitute nibbles Instead of dividing the block into a four by four array of bytes, S-AES divides it into a two by two array of –nibbles‖, which are four bits long. This is called the state array and is shown below.



S-AES state array

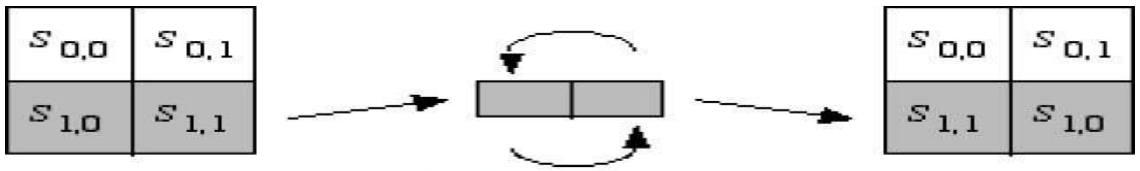
In the first stage of each encryption round, an S-box is used to translate each nibble into a new nibble. First we associate the nibble b0b1b2b3 with the polynomial b0x3 + b1x2 + b2x + b3. This polynomial is then inverted as an element of GF(16), with the –prime polynomial‖ used being.x4 + x + 1. Then we multiply by a matrix and add a vector as in AES.

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Remember that the addition and multiplication in the equation above are being done modulo 2 (with XOR), but not in GF(16). Since a computer would do the S-box substitution using a table lookup, we give the full table for the S-box here.

nibble	S-box(nibble)	nibble	S-box(nibble)
0000	1001	1000	0110
0001	0100	1001	0010
0010	1010	1010	0000
0011	1011	1011	0011
0100	1101	1100	1100
0101	0001	1101	1110
0110	1000	1110	1111
0111	0101	1111	0111

Shift Rows The next stage is to shift the rows. In fact, the first row is left alone and the second row is shifted.



S-AES shift row transformation

Mix Columns After shifting the rows, we mix the columns. Each column is multiplied by the Matrix

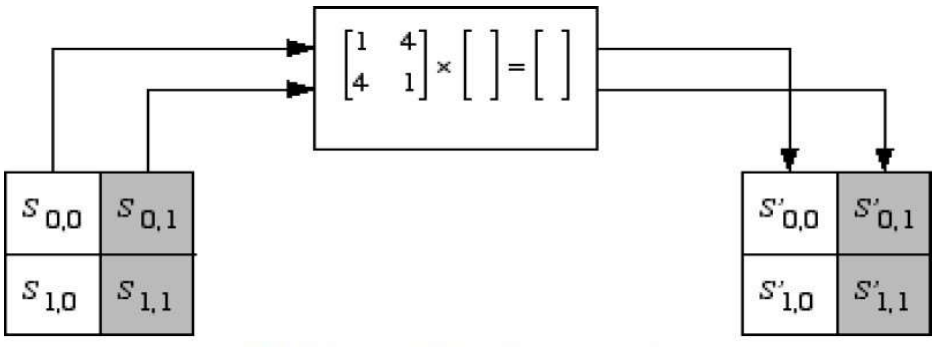
$$\begin{bmatrix} 1 & 4 \\ 4 & 1 \end{bmatrix}.$$

These operations are done in GF(16), so remember that the 1 corresponds to the polynomial 1 and the 4 corresponds to the polynomial x2. Thus this matrix could also be written as

$$\begin{bmatrix} 1 & x^2 \\ x^2 & 1 \end{bmatrix}.$$

Don‘t forget to reduce modulo x4 + x + 1.

The mix column transformation is omitted in the last round, in order to simplify the decryption.

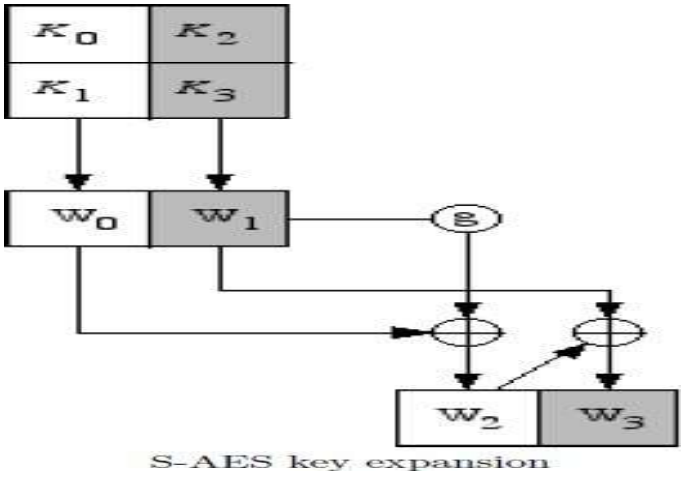


S-AES mix column transformation

Add Round Key The last stage of each round of encryption is to add the round key. (In fact, this is also done before the first round.) Before the first round, the first two words (W0 and W1) of the expanded key are added. In the first round, W2 and W3 are added. In the last round, W4 and W5 are added. All additions are done modulo 2, that is, with XOR.

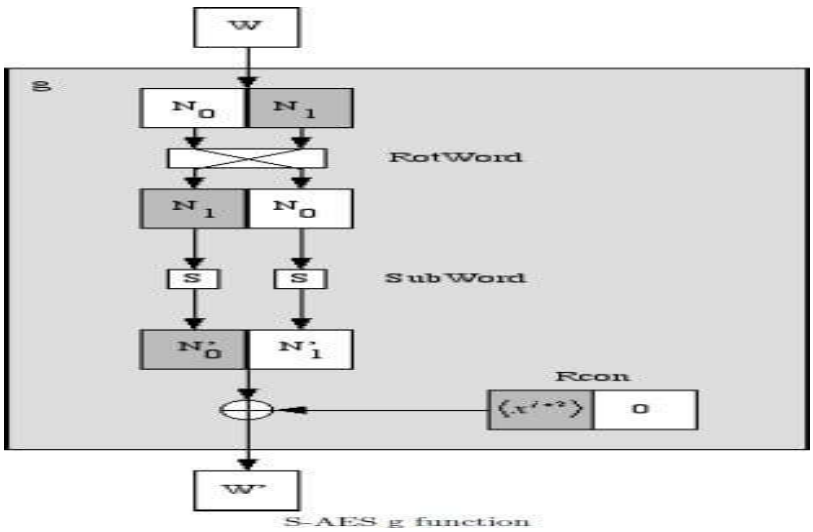
Key Expansion Key expansion is done very similarly to AES. The four nibbles in the key are grouped into two 8-bit

–words‖, which will be expanded into 6 words. The first part of the expansion, which produces the third and fourth words, is shown below. The rest of the expansion is done in exactly the same way, replacing W0 and W1 with W2 and W3, and replacing W2 and W3 with W4 and W5.



S-AES key expansion

The g function is shown in the next diagram. It is very similar to AES, first rotating the nibbles and then putting them through the S-boxes. The main difference is that the round constant is produced using xj+2, where j is the number of the round of expansion. That is, the first time you expand the key you use a round constant of x3 = 1000 for the first nibble and 0000 for the



S-AES g function

second nibble. The second time you use x4 = 0011 for the first nibble and 0000 for the second nibble.

2. SOLVED EXAMPLE OF S-AES

(Note : Assignment for students)

ALGORITHM

The gist of AES is this: we arrange each block of the plaintext into a 4x4 matrix and repeatedly perform a set of operations on it. We call each iteration a round, and we perform 10, 12 or 14 rounds depending on the key length (this is another parameter chosen by NIST):

- 10 rounds for a 128-bit key
- 12 rounds for a 196-bit key
- 14 rounds for a 256 bit key

For each round, we generate a round key from the main key using the Rijndael Key Schedule.

There are four operations on the 4x4 matrix that we will define:

- subBytes()
- shiftRows()
- mixColumns()
- addRoundKey()

Not every round of operations is the same; for the first round, we only add the round key, and for the last round we omit the mixColumns() step. So the pseudocode for the AES algorithm might look something like this:

```
function AESencrypt(plaintext, key)
{
 blocks := divideIntoBlocks(plaintext);
 roundKeys = getRoundKeys(key)  for (block in blocks) {    //first round
   addRoundKey(roundKeys[0], block);    //intermediate rounds
   for (8, 10 or 12 rounds)
 {
     subBytes(block);
     shiftRows(block);
     mixColumns(block);
     addRoundKey(roundKeys[..], block);
   }    //last round
   subBytes(block);
   shiftRows(block);
   addRoundKey(roundKeys[numRounds - 1], block);  }  ciphertext := reassemble(blocks);
 return ciphertext;}
```

CONCLUSION: Thus the implementation of S-AES algorithm was successfully conducted.

ASSIGNMENT NO. 03

AIM: To implement a Simplified Data Encryption Standard (S-DES) algorithm.

OBJECTIVES:

- The Basic Concepts of S-DES.
- General structure of S-DES.
- Logical implementation of S-DES.

THEORY:

1. INTRODUCTION

Figure G.1 illustrates the overall structure of the Simplified DES, which we will refer to as S-DES. The S-DES encryption algorithm takes an 8-bit block of plaintext (example: 10111101) and a 10-bit key as input and produces an 8-bit block of cipher text as output. The S-DES decryption algorithm takes an 8-bit block of cipher text and the same 10-bit key used to produce that cipher text as input and produces the original 8-bit block of plaintext. The encryption algorithm involves five functions: an initial permutation (IP); a complex function labelled f*K*, which involves both permutation and substitution operations and depends on a key input; a simple permutation function that switches (SW) the two halves of the data; the function f*K* again; and finally a permutation function that is the inverse of the initial permutation (IP–1).

The function f*K* takes as input not only the data passing through the encryption algorithm, but also an 8-bit key. The algorithm could have been designed to work with a 16-bit key, consisting of two 8-bit subkeys, one used for each occurrence of f*K*. Alternatively, a single 8-bit key could have been used, with the same key used twice in the algorithm. A compromise is to use a 10-bit key from which two 8-bit subkeys are generated, as depicted in Figure G.1. In this case, the key is first subjected to a permutation (P10). Then a shift operation is performed. The output of the shift operation then passes through a permutation function that reduces an 8-bit output (P8) for the first subkey (*K*1). The output of the shift operation also feeds into another shift and another instance of P8 to produce the second subkey (*K*2).

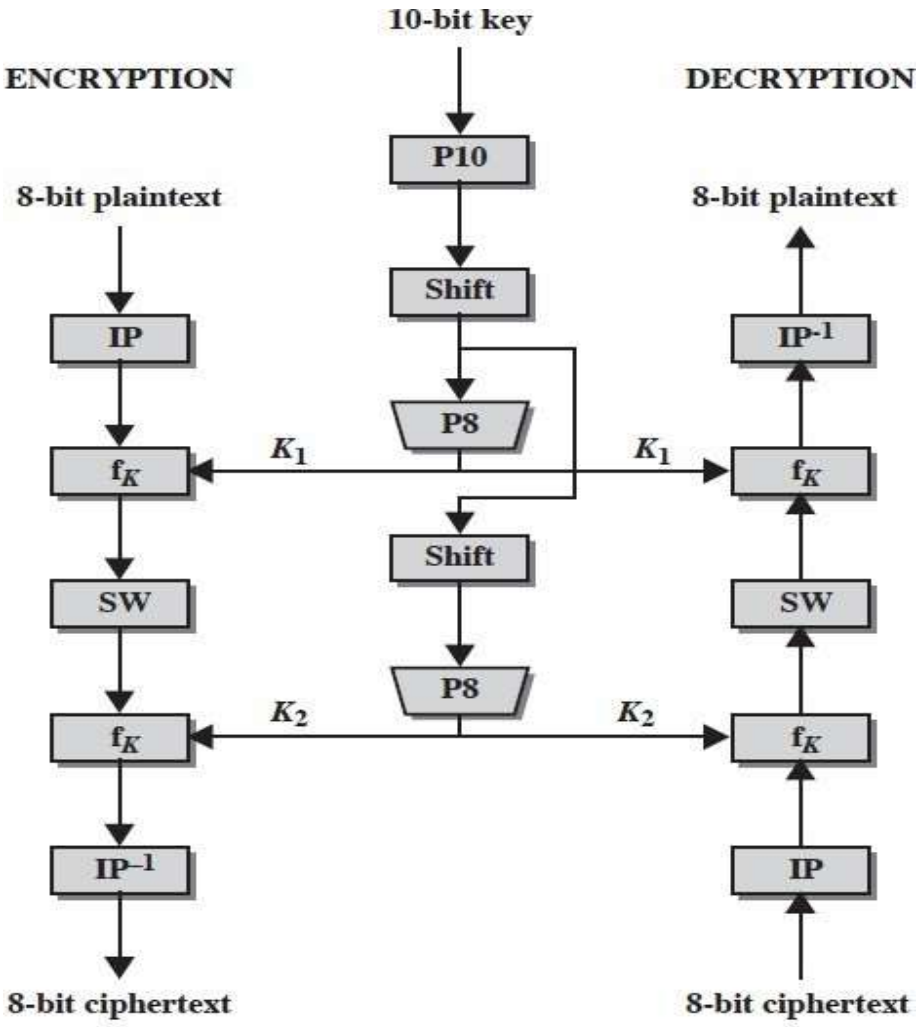


Figure G.1 Simplified DES Scheme

We can concisely express the encryption algorithm as a composition of functions:

$$\mathrm{IP}^{-1} \circ \mathrm{f}_{K_2} \circ \mathrm{SW} \circ \mathrm{f}_{K_1} \circ \mathrm{IP}$$

which can also be written as:

$$\text{ciphertext} = \mathrm{IP}^{-1}\left(\mathrm{f}_{K_2}\left(\mathrm{SW}\left(\mathrm{f}_{K_1}\left(\mathrm{IP}(\text{plaintext})\right)\right)\right)\right)$$

Where

$$K_1 = \mathrm{P8}\left(\mathrm{Shift}\left(\mathrm{P10}(\text{key})\right)\right)$$

$$K_2 = \mathrm{P8}\left(\mathrm{Shift}\left(\mathrm{Shift}\left(\mathrm{P10}(\mathrm{key})\right)\right)\right)$$

Decryption is also shown in Figure G.1 and is essentially the reverse of encryption:

$$\text{plaintext} = \mathrm{IP}^{-1}\left(\mathrm{f}_{K_1}\left(\mathrm{SW}\left(\mathrm{f}_{K_2}\left(\mathrm{IP}(\text{ciphertext})\right)\right)\right)\right)$$

2. S-DES KEY GENERATION

S-DES depends on the use of a 10-bit key shared between sender and receiver. From this key, two 8-bit subkeys are produced for use in particular stages of the encryption and decryption algorithm. Figure G.2 depicts the stages followed to produce the subkeys. First, permute the key in the following fashion. Let the 10-bit key be designated as (*k*1, *k*2, *k*3, *k*4, *k*5, *k*6, *k*7, *k*8, *k*9, *k*10). Then the permutation P10 is defined

$$\mathrm{P10}(k_1, k_2, k_3, k_4, k_5, k_6, k_7, k_8, k_9, k_{10}) = (k_3, k_5, k_2, k_7, k_4, k_{10}, k_1, k_9, k_8, k_6)$$

as:

P10 can be concisely defined by the display:

P10									
3	5	2	7	4	10	1	9	8	6

This table is read from left to right; each position in the table gives the identity of the input bit that produces the output bit in that position. So the first output bit is bit 3 of the input; the second output bit is bit 5 of the input, and so on. For example, the key (1010000010) is permuted to (1000001100). Next, perform a circular left shift (LS-1), or rotation, separately on the first five bits and the second five bits. In our example, the result is (00001 11000).

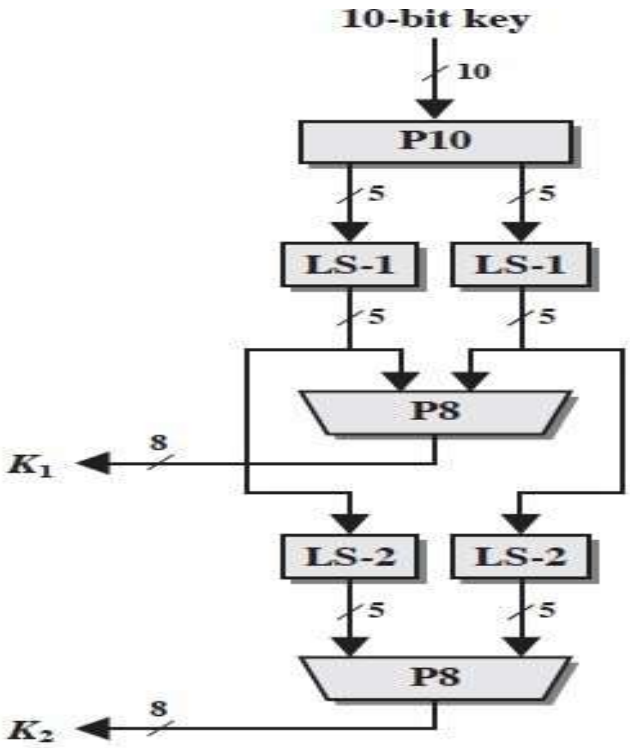


Figure G.2 Key Generation for Simplified DES

Next we apply P8, which picks out and permutes 8 of the 10 bits according to the following rule:

P8							
6	3	7	4	8	5	10	9

The result is subkey 1 (*K*1). In our example, this yields (10100100)

We then go back to the pair of 5-bit strings produced by the two LS-1 functions and perform a circular left shift of 2 bit positions on each string. In our example, the value (0000111000) becomes (00100 00011). Finally, P8 is applied again to produce *K*2. In our example, the result is (01000011).

Figure G.3 shows the S-DES encryption algorithm in greater detail.

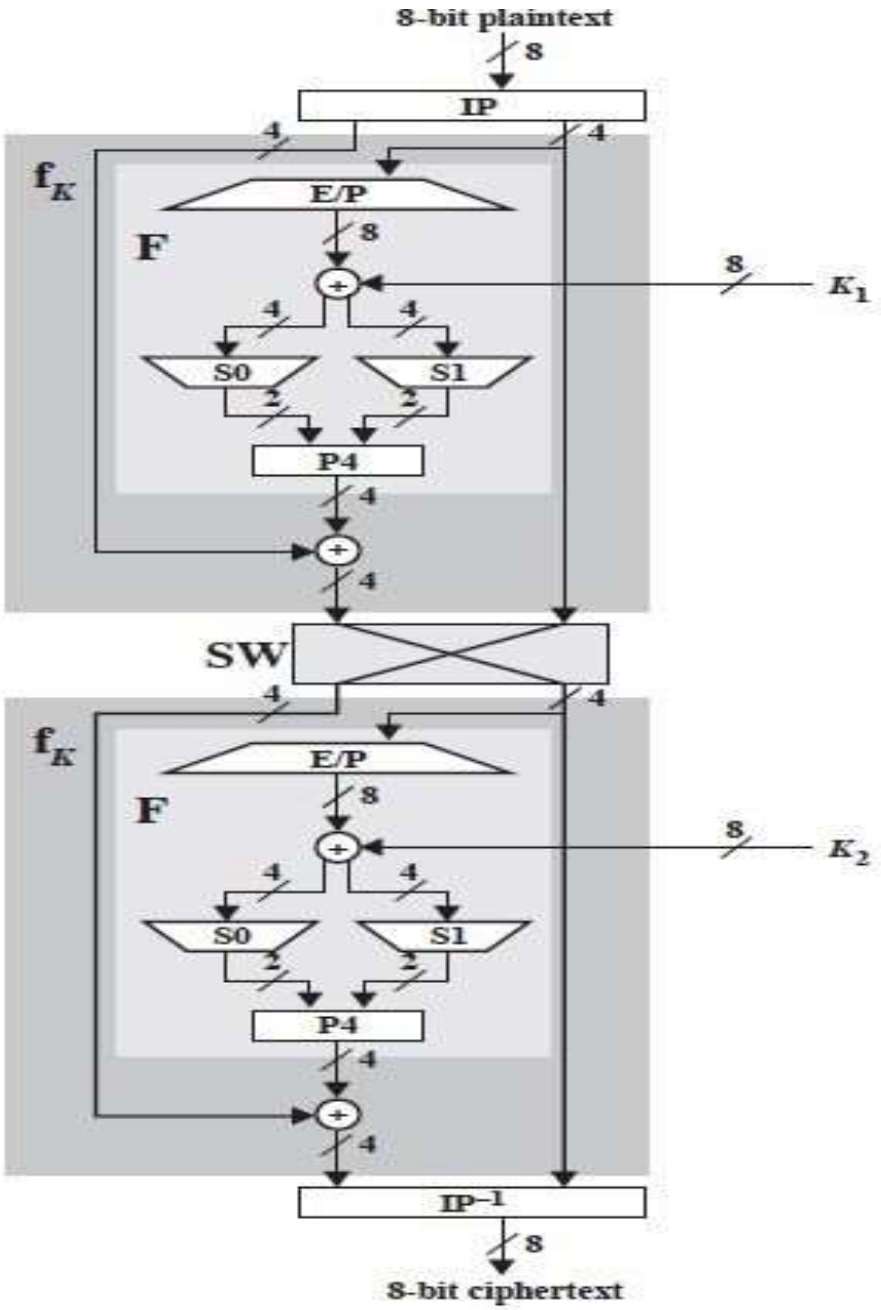


Figure G.3 Simplified DES Encryption Detail

3. FUNCTIONS

As it was mentioned, encryption involves the sequential application of five functions. We examine each of these.

a. Initial and Final Permutations

The input to the algorithm is an 8-bit block of plaintext, which we first permute using the IP function:

IP							
2	6	3	1	4	8	5	7

This retains all 8 bits of the plaintext but mixes them up. At the end of the algorithm, the inverse permutation is used:

IP^{-1}							
4	1	3	5	7	2	8	6

It is easy to show by example that the second permutation is indeed the reverse of the first, that is

$$IP^{-1}(IP(X)) = X.$$

The Function f*K*

The most complex component of S-DES is the function f*K*, which consists of a combination of permutation and substitution functions. The functions can be expressed as follows. Let *L* and *R* be the leftmost 4 bits and rightmost 4 bits of the 8-bit input to f*K*, and let F be a mapping (not necessarily one to one) from 4-bit strings to 4-bit strings. Then we let

$$f_K(L, R) = (L \oplus F(R, SK), R)$$

where *SK* is a subkey and ! is the bit-by-bit exclusive-OR function. For example, suppose the output of the IP stage in Figure G.3 is (10111101) and F(1101, *SK*) = (1110) for some key *SK*. Then f*K*(10111101) = (01011101) because (1011) ⊕ (1110) = (0101).

We now describe the mapping F. The input is a 4-bit number (*n*1*n*2*n*3*n*4). The first operation is an expansion/permutation operation:

E/P							
4	1	2	3	2	3	4	1

For what follows, it is clearer to depict the result in this fashion:

$$\begin{array}{c|cc|c} n_4 & n_1 & n_2 & n_3 \\ n_2 & n_3 & n_4 & n_1 \end{array}$$

The 8-bit subkey $K1 = (k11, k12, k13, k14, k15, k16, k17, k18)$ is added to this value using exclusive-OR:

$$\begin{array}{c|cc|c} n_4 \oplus k_{11} & n_1 \oplus k_{12} & n_2 \oplus k_{13} & n_3 \oplus k_{14} \\ n_2 \oplus k_{15} & n_3 \oplus k_{16} & n_4 \oplus k_{17} & n_1 \oplus k_{18} \end{array}$$

Let us rename these 8 bits:

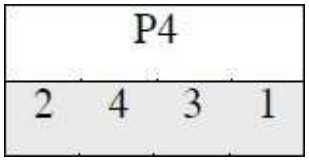
$$\begin{array}{c|cc|c} p_{0,0} & p_{0,1} & p_{0,2} & p_{0,3} \\ p_{1,0} & p_{1,1} & p_{1,2} & p_{1,3} \end{array}$$

The first 4 bits (first row of the preceding matrix) are fed into the S-box S0 to produce a 2- bit output, and the remaining 4 bits (second row) are fed into S1 to produce another 2-bit output. These two boxes are defined as follows:

$$S0 = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 1 & 0 & 3 & 2 \\ 3 & 2 & 1 & 0 \\ 0 & 2 & 1 & 3 \\ 3 & 1 & 3 & 2 \end{bmatrix} \end{array} \qquad S1 = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 1 & 2 & 3 \\ 2 & 0 & 1 & 3 \\ 3 & 0 & 1 & 0 \\ 2 & 1 & 0 & 3 \end{bmatrix} \end{array}$$

The S-boxes operate as follows. The first and fourth input bits are treated as a 2-bit number that specify a row of the S-box, and the second and third input bits specify a column of the Sbox. The entry in that row and column, in base 2, is the 2-bit output. For example, if $(p0,0p0,3) = (00)$ and $(p0,1p0,2) = (10)$, then the output is from row 0, column 2 of S0, which is 3, or (11) in binary. Similarly, $(p1,0p1,3)$ and $(p1,1p1,2)$ are used to index into a row and column of S1 to produce an additional 2 bits.

Next, the 4 bits produced by S0 and S1 undergo a further permutation as follows:



The output of P4 is the output of the function F.

b. The Switch Function

The function f*K* only alters the leftmost 4 bits of the input. The switch function (SW) interchanges the left and right 4 bits so that the second instance of f*K* operates on a different 4 bits. In this second instance, the E/P, S0, S1, and P4 functions are the same. The key input is *K*2.

SOLVED EXAMPLE OF S-DES

(Note : Assignment for students)

ALGORITHM

Pseudocode for DES cipher

```
Cipher (plainBlock[64], RoundKeys[16, 48], cipherBlock[64])
{
 permute (64, 64, plainBlock, inBlock, InitialPermutationTable)
 split (64, 32, inBlock, leftBlock, rightBlock)
 for (round = 1 to 16)
 {
 mixer (leftBlock, rightBlock, RoundKeys[round])
 if (round!=16) swapper (leftBlock, rightBlock)
 }
 combine (32, 64, leftBlock, rightBlock, outBlock)
 permute (64, 64, outBlock, cipherBlock, FinalPermutationTable)
}
mixer (leftBlock[48], rightBlock[48], RoundKey[48])
{
 copy (32, rightBlock, T1)
```

```
 function (T1, RoundKey, T2)
 exclusiveOr (32, leftBlock, T2, T3)
 copy (32, T3, rightBlock)
}
swapper (leftBlock[32], rigthBlock[32])
{
 copy (32, leftBlock, T)
 copy (32, rightBlock, leftBlock)
 copy (32, T, rightBlock)
}
function (inBlock[32], RoundKey[48], outBlock[32])
{
 permute (32, 48, inBlock, T1, ExpansionPermutationTable)
 exclusiveOr (48, T1, RoundKey, T2)
 substitute (T2, T3, SubstituteTables)
 permute (32, 32, T3, outBlock, StraightPermutationTable)
}
substitute (inBlock[32], outBlock[48], SubstitutionTables[8, 4, 16])
{
 for (i = 1 to 8)
 {
 row ¨ 2 ¥ inBlock[i ¥ 6 + 1] + inBlock [i ¥ 6 + 6]
 col ¨ 8 ¥ inBlock[i ¥ 6 + 2] + 4 ¥ inBlock[i ¥ 6 + 3] +
 2 ¥ inBlock[i ¥ 6 + 4] + inBlock[i ¥ 6 + 5]
 value = SubstitutionTables [i][row][col]
 outBlock[[i ¥ 4 + 1] ¨ value / 8; value ¨ value mod 8
 outBlock[[i ¥ 4 + 2] ¨ value / 4; value ¨ value mod 4
 outBlock[[i ¥ 4 + 3] ¨ value / 2; value ¨ value mod 2
 outBlock[[i ¥ 4 + 4] ¨ value
 }}
```

CONCLUSION: Thus the implementation of S-DES algorithm was successfully conducted.

ASSIGNMENT NO. 05

AIM: To implement a RSA algorithm.

OBJECTIVES: Students will learn:

- The Basic Concepts of RSA.
- General structure of RSA.
- Logical implementation of RSA.

THEORY:

1. INTRODUCTION

RSA (**Rivest–Shamir–Adleman**) is one of the first public-key cryptosystems and is widely used for secure data transmission. It is an asymmetric cryptographic algorithm. Asymmetric means that there are two different keys. This is also called public key cryptography, because one of them can be given to everyone. In such a cryptosystem, the encryption key is public and it is different from the decryption key which is kept secret (private). In RSA, this asymmetry is based on the practical difficulty of the factorization of the product of two large prime numbers, the "factoring problem". The acronym RSA is made of the initial letters of the surnames of Ron Rivest, Adi Shamir, and Leonard Adleman, who first publicly described the algorithm in 1978.

A user of RSA creates and then publishes a public key based on two large prime numbers, along with an auxiliary value. The prime numbers must be kept secret. Anyone can use the public key to encrypt a message, but with currently published methods, and if the public key is large enough, only someone with knowledge of the prime numbers can decode the message feasibly. Breaking RSA encryption is known as the RSA problem.

RSA is a relatively slow algorithm, and because of this, it is less commonly used to directly encrypt user data. More often, RSA passes encrypted shared keys for symmetric key cryptography which in turn can perform bulk encryption-decryption operations at much higher speed.

2. OPERATIONS

The RSA algorithm involves four steps: **key generation, key distribution, encryption and decryption.**

A basic principle behind RSA is the observation that it is practical to find three very large positive integers *e*, *d* and *n* such that with modular exponentiation for all integers *m* (with $0 \leq m < n$):

$$(m^e)^d = m \pmod{n}$$

and that even knowing *e* and *n* or even *m* it can be extremely difficult to find *d*.

In addition, for some operations it is convenient that the order of the two exponentiations can be changed and that this relation also implies:

$$(m^d)^e = m \pmod n$$

RSA involves a *public key* and a *private key.* The public key can be known by everyone, and it is used for encrypting messages. The intention is that messages encrypted with the public key can only be decrypted in a reasonable amount of time by using the private key. The public key is represented by the integers n and e; and, the private key, by the integer d(although n is also used during the decryption process. Thus, it might be considered to be a part of the private key, too). m represents the message.

a. Key generation

The keys for the RSA algorithm are generated the following way:

1. Choose two distinct prime numbers p and q.

   For security purposes, the integers p and q should be chosen at random, and should be similar in magnitude but differ in length by a few digits to make factoring harder. Prime integers can be efficiently found using a primarily test.

2. Compute $n = pq$.

   n is used as the modulus for both the public and private keys. Its length, usually expressed in bits, is the key length.

3. Compute $\lambda(n) = \text{lcm}(\lambda(p), \lambda(q)) = \text{lcm}(p - 1, q - 1)$, where λ is Carmichael's totient function. This value is kept private.
4. Choose an integer e such that $1 < e < \lambda(n)$ and $\gcd(e, \lambda(n)) = 1$; i.e., e and $\lambda(n)$ are co-prime.
5. Determine d as $d \equiv e^{-1} \pmod{\lambda(n)}$; i.e., d is the modular multiplicative inverse of e modulo $\lambda(n)$.
   - This means: solve for d the equation $d \cdot e \equiv 1 \pmod{\lambda(n)}$.
   - e having a short bit-length and small Hamming weight results in more efficient encryption – most commonly $e = 2^{16} + 1 = 65{,}537$. However, much smaller values of e (such as 3) have been shown to be less secure in some settings.
   - e is released as the public key exponent.
   - d is kept as the private key exponent.

The *public key* consists of the modulus n and the public (or encryption) exponent e. The *private key* consists of the private (or decryption) exponent d, which must be kept secret. p, q, and $\lambda(n)$ must also be kept secret because they can be used to calculate d.

b. Key distribution

Suppose that Bob wants to send information to Alice. If they decide to use RSA, Bob must know

Alice's public key to encrypt the message and Alice must use her private key to decrypt the message. To enable Bob to send his encrypted messages, Alice transmits her public key (n, e) to Bob via a reliable, but not necessarily secret, route. Alice's private key (d) is never distributed.

c. Encryption

After Bob obtains Alice's public key, he can send a message M to Alice. To do it, he first turns M (strictly speaking, the un-padded plaintext) into an integer m (strictly speaking, the padded plaintext), such that $0 \leq m < n$ by using an agreed-upon reversible protocol known as a padding scheme. He then computes the cipher text c, using Alice's public key e, corresponding to

$$c \equiv m^e \pmod{n}$$

Bob then transmits c to Alice.

d. Decryption

Alice can recover m from c by using her private key exponent d by computing

$$m \equiv c^d \pmod{n}$$

Given m, she can recover the original message M by reversing the padding scheme.

3. STEPS OF RSA ALGORITHM

STEP-1: Select two co-prime numbers as p and q.

STEP-2: Compute n as the product of p and q.

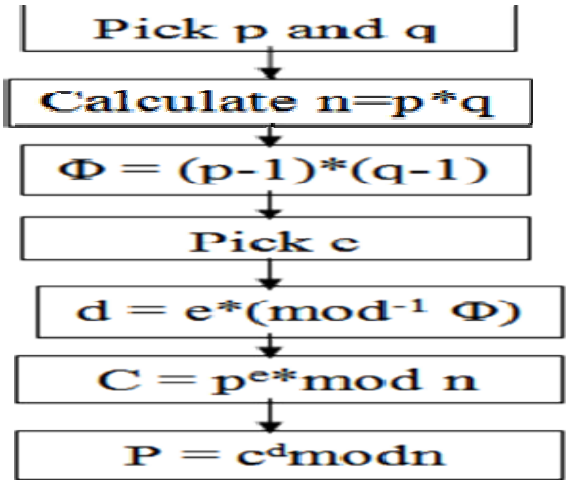
STEP-3: Compute (p-1)*(q-1) and store it in z.

STEP-4: Select a random prime number e that is less than that of z.

STEP-5: Compute the private key, d as e * mod^{-1} (z).

STEP-6: The cipher text is computed as $message^{e}$ * mod n.

STEP-7: Decryption is done as $cipher^{d}$ * mod n.



4. EXAMPLE OF RSA ALGORITHM

1. Select primes: p = 17 ; q = 11
2. Calculate n = pq = 17 x 11 = 187
3. Calculate ø(n) = (p–1)(q-1) = 16x10 = 160
4. Select e: GCD(e,160) = 1 ; choose e = 7
5. Derive d: de = 1 mod 160 and d < 160

Get d = 23 since 23x7 = 161 = 10x160+1

6. Publish public key: PU = {7,187}
7. Keep private key secret: PR = {23,187}

RSA Encryption and Decryption :

- Sample RSA encryption/decryption is:
- given message M = 88 (nb 88 < 187)
- Encryption:

$$C = 88^7 \bmod 187 = 11$$

- Decryption:

$$M = 11^{23} \bmod 187 = 88$$

CONCLUSION: Thus the implementation of RSA algorithm was successfully conducted.

ASSIGNMENT NO. 06

AIM: To implement a Diffie-Hellman Key Exchange algorithm.

OBJECTIVES:

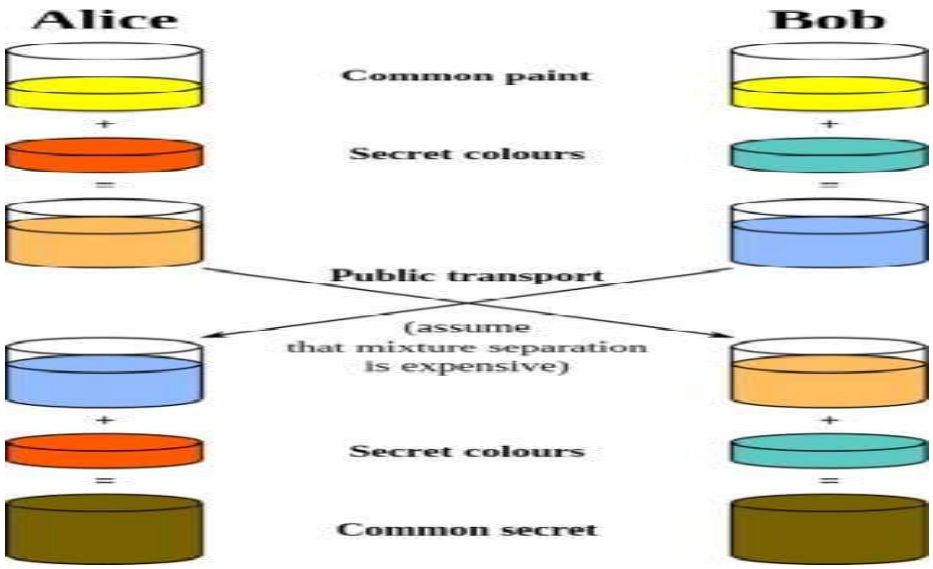
- The Basic Concepts of a Diffie-Hellman key exchange.
- General structure Diffie-Hellman key exchange.
- Logical implementation of Diffie-Hellman key exchange.

THEORY

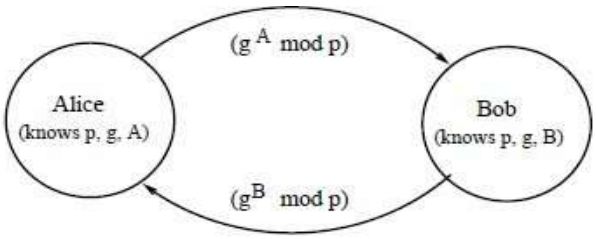
1. INTRODUCTION

Diffie–Hellman key exchange (**DH**) is a method of securely exchanging cryptographic keys over a public channel and was one of the first public-key protocols as originally conceptualized by Ralph Merkle and named after Whitfield Diffie and Martin Hellman DH is a mathematical algorithm that permits two PCs to produce an identical shared secret on both systems, despite the fact that those systems might never have communicated with one another. That shared secret can then be utilized to safely exchange a cryptographic encryption key. That key then encrypts traffic between the two systems. **Diffie-Hellman is not an encryption mechanism that is we don't commonly utilize DH to encrypt data. Rather, it is a strategy for secure exchange of the keys that encrypt data.**

The following diagram shows the general thought of the key exchange by utilizing colours rather than a large number. The procedure starts by having the two gatherings, Alice and Bob, agree on an arbitrary starting colour that does not should be kept secret; in this example the colour is yellow. Each of them chooses a secret colour - red and aqua respectively - that they keep to themselves. The vital piece of the procedure is that Alice and Bob now combine their secret colour with their commonly shared colour, bringing about orange and blue mixtures respectively, and then freely exchange the two mixed colours. At last, each of the two combine the colour they got from the partner with their own particular private colour. The outcome is a last colour mixture (brown) that is identical with the partner's colour mixture.



2. STEPS OF DEFFIE HELLMAN KEY EXCHANGE



Steps in the algorithm:

1. Alice and Bob agree on a prime number p and a base g.
2. Alice chooses a secret number a, and sends Bob $(g^a \mod p)$.
3. Bob chooses a secret number b, and sends Alice $(g^b \mod p)$.
4. Alice computes $((g^b \mod p)^a \mod p)$.
5. Bob computes $((g^a \mod p)^b \mod p)$.

Both Alice and Bob can use this number as their key. Notice that p and g need not be protected.

3. EXAMPLE OF DEFFIE HELLMAN KEY EXCHANGE

1. g = public (prime) base, known to Alice, Bob, and Eve.

   g = 5
2. p = public (prime) modulus, known to Alice, Bob, and Eve.

   p = 23
3. a = Alice's private key, known only to Alice.

   a = 6
4. b = Bob's private key known only to Bob.

   b = 15
5. A = Alice's public key, known to Alice, Bob, and Eve.

   $A = g^a \bmod p = 8$
6. B = Bob's public key, known to Alice, Bob, and Eve.

   $B = g^b \bmod p = 19$
7. Alice computes $19^6 \bmod 23 = 2$
8. Bob computes $8^{15} \bmod 23 = 2$
9. Then 2 is the shared secret.

   Above example is pictorially represented as follows :

Alice		Bob		Eve	
Known	Unknown	Known	Unknown	Known	Unknown
$p = 23$	b	$p = 23$	a	$p = 23$	a
$g = 5$		$g = 5$		$g = 5$	b
$a = 6$		$b = 15$			s
$A = 5^a \bmod 23$		$B = 5^b \bmod 23$		$A = 8$	
$A = 5^6 \bmod 23 = 8$		$B = 5^{15} \bmod 23 = 19$		$B = 19$	
$B = 19$		$A = 8$		$s = 19^a \bmod 23 = 8^b \bmod 23$	
$s = B^a \bmod 23$		$s = A^b \bmod 23$			
$s = 19^6 \bmod 23 = 2$		$s = 8^{15} \bmod 23 = 2$			
$s = 2$		$s = 2$			

4. SECURITY ISSUE

Man-In-The-Middle Attack:

This algorithm has a noteworthy weakness as man - in - the - middle vulnerability. In this attack, a malicious third party, usually referred to as –Eve‖ (for –eavesdropper‖) recovers Alice‘s public key and sends her own public key to Bob. At the point when Bob transmits his public key, Eve interrupts on and substitutes the value with her own public key and after that sends it to Alice. At this point, Alice would have gone to an agreement on a common secret key with Eve rather than Bob and it is feasible for Eve to decrypt any messages conveyed by Alice or Bob, and after that read and perhaps alter them before the re- encryption with the suitable key and transmitting them to alternate party.

This weakness is available on the grounds that Diffie Hellman key exchange does not authenticate the members. Possible solutions include the use of digital signatures and other protocol variants.

5. ADVANTAGES

1. The security factors with respect to the fact that solving the discrete logarithm is very challenging.
2. That the shared key (i.e. the secret) is never itself transmitted over the channel.

6. DISADVANTAGES

1. The fact that there are expensive exponential operations involved, and the algorithm cannot be used to encrypt messages - it can be used for establishing a secret key only.
2. There is also a lack of authentication.
3. There is no identity of the parties involved in the exchange.
4. It is easily susceptible to man-in-the-middle attacks. A third party C, can exchange keys with both A and B, and can listen to the communication between A and B.
5. The algorithm is computationally intensive. Each multiplication varies as the square of n, which must be very large. The number of multiplications required by the exponentiation increases with increasing values of the exponent, x or y in this case.
6. The computational nature of the algorithm could be used in a denial of-service attack very easily.

ALGORITHM

Step 1 : Choose two prime numbers g(primitive root of p) and **p.**

Step 2 : Alice selects a secret no(**a**) and computes **ga** *mod* **p** , let's call it **A**. Alice sends **A** to Bob.

Step 3 : Bob selects a secret no(**b**) and computes **gb** *mod* **p,** let's call it **B.** Bob sends **B** to Alice.

Step 4 : Alice computes S_A = Ba mod p

Step 5 : Bob computes S_B = Ab mod p

Step 6 : If **S_A=S_B** then Alice and Bob can agree for future communication.

CONCLUSION: Thus the implementation of Diffie-Hellman key exchange algorithm was successfully conducted.

ASSIGNMENT NO. 07

AIM: Calculate the message digest of a text using the MD5 algorithm in JAVA

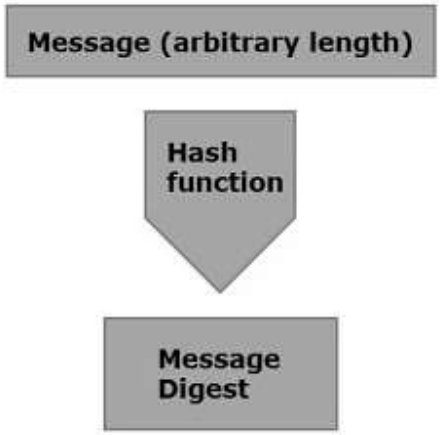
OBJECTIVES:

THEORY:

Hash functions are extremely useful and appear in almost all information security applications.

A hash function is a mathematical function that converts a numerical input value into another compressed numerical value. The input to the hash function is of arbitrary length but output is always of fixed length.

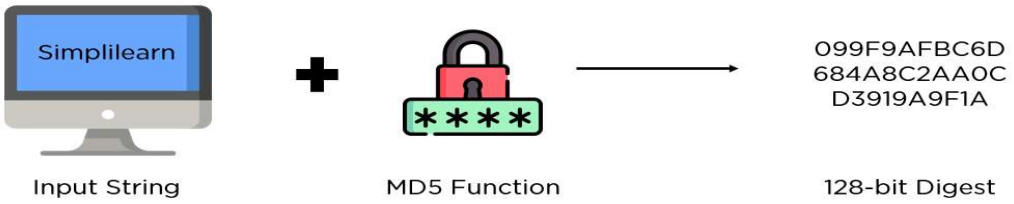
Values returned by a hash function are called **message digest** or simply **hash values**. The following picture illustrated hash function.



Java provides a class named **MessageDigest** which belongs to the package java.security. This class supports algorithms such as SHA-1, SHA 256, MD5 algorithms to convert an arbitrary length message to a message digest.

What is the MD5 Algorithm?

MD5 (Message Digest Method 5) is a cryptographic hash algorithm used to generate a 128-bit digest from a string of any length. It represents the digests as 32 digit hexadecimal numbers.



Ronald Rivest designed this algorithm in 1991 to provide the means for digital signature verification. Eventually, it was integrated into multiple other frameworks to bolster security indexes.

Input into Hash Function	MD5 Hash Value/Digest
Cryptography	d2fc0657a64a3291826136c7712abbe7
Cryptographyabc123	c56db83ab5482b4e94536f4a29b21de0
Cryptographyxyz456	783b10b483435e05f3f2705bdd5a825c

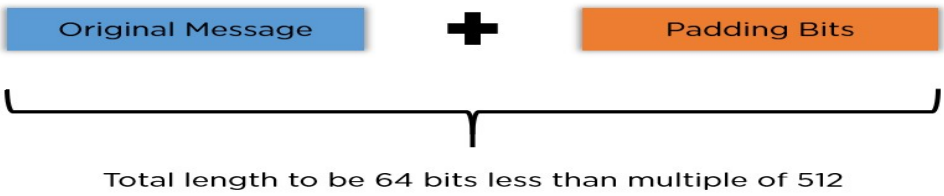
The digest size is always 128 bits, and thanks to hashing function guidelines, a minor change in the input string generate a drastically different digest. This is essential to prevent similar hash generation as much as possible, also known as a hash collision.

MD5 ALGORITHM :

There are four major sections of the algorithm:

Padding Bits

When you receive the input string, you have to make sure the size is 64 bits short of a multiple of 512. When it comes to padding the bits, you must add one(1) first, followed by zeroes to round out the extra characters.



Padding Length

You need to add a few more characters to make your final string a multiple of 512. To do so, take the length of the initial input and express it in the form of 64 bits. On combining the two, the final string is ready to be hashed.

Initialize MD Buffer

The entire string is converted into multiple blocks of 512 bits each. You also need to initialize four different buffers, namely A, B, C, and D. These buffers are 32 bits each and are initialized as follows:

A = 01 23 45 67

B = 89 ab cd ef

C = fe dc ba 98

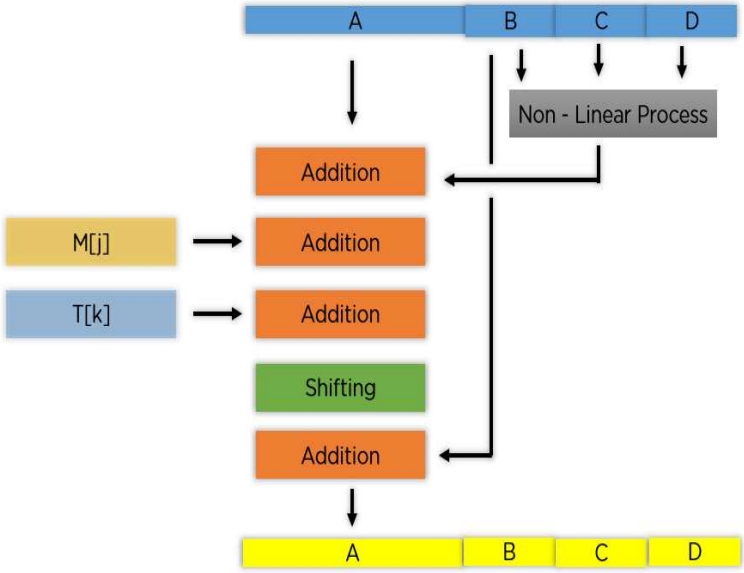
D = 76 54 32 10

Process Each Block

Each 512-bit block gets broken down further into 16 sub-blocks of 32 bits each. There are four rounds of operations, with each round utilizing all the sub-blocks, the buffers, and a constant array value.

This constant array can be denoted as T[1] -> T[64].

Each of the sub-blocks are denoted as M[0] -> M[15].



According to the image above, you see the values being run for a single buffer A. The correct order is as follows:

- It passes B, C, and D onto a non-linear process.
- The result is added with the value present at A.
- It adds the sub-block value to the result above.
- Then, it adds the constant value for that particular iteration.
- There is a circular shift applied to the string.
- As a final step, it adds the value of B to the string and is stored in buffer A.

The steps mentioned above are run for every buffer and every sub-block. When the last block's final buffer is complete, you will receive the MD5 digest.

The non-linear process above is different for each round of the sub-block.

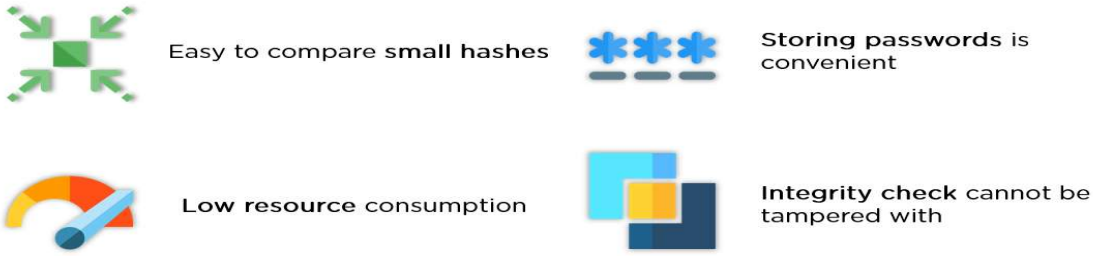
Round 1: (b AND c) OR ((NOT b) AND (d))

Round 2: (b AND d) OR (c AND (NOT d))

Round 3: b XOR c XOR d

Round 4: c XOR (b OR (NOT d))

Advantages of MD5



- Easy to Compare: Unlike the latest hash algorithm families, a 32 digit digest is relatively easier to compare when verifying the digests.
- Storing Passwords: Passwords need not be stored in plaintext format, making them accessible for hackers and malicious actors. When using digests, the database also gets a boost since the size of all hash values will be the same.
- Low Resource: A relatively low memory footprint is necessary to integrate multiple services into the same framework without a CPU overhead.
- Integrity Check: You can monitor file corruption by comparing hash values before and after transit. Once the hashes match, file integrity checks are valid, and it avoids data corruption.

To convert a given message to a message digest, follow the steps given below −

Step 1: Create a MessageDigest object

The MessageDigest class provides a method named **getInstance()**. This method accepts a String variable specifying the name of the algorithm to be used and returns a MessageDigest object implementing the specified algorithm.

Create MessageDigest object using the **getInstance()** method as shown below.

```
MessageDigest md = MessageDigest.getInstance("SHA-256");
```

Step 2: Pass data to the created MessageDigest object

After creating the message digest object, you need to pass the message/data to it. You can do so using the **update()** method of the **MessageDigest** class, this method accepts a byte array representing the message and adds/passes it to the above created MessageDigest object.

```
md.update(msg.getBytes());
```

Step 3: Generate the message digest

You can generate the message digest using the **digest()** method od the MessageDigest class this method computes the hash function on the current object and returns the message digest in the form of byte array.

Generate the message digest using the digest method.

```
byte[] digest = md.digest();
```

Example

Following is an example which reads data from a file and generate a message digest and prints it.

```
import java.security.MessageDigest;
import java.util.Scanner;

public class MessageDigestExample {
  public static void main(String args[]) throws Exception{
    //Reading data from user
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter the message");
    String message = sc.nextLine();

    //Creating the MessageDigest object
    MessageDigest md = MessageDigest.getInstance("MD5");

    //Passing data to the created MessageDigest Object
    md.update(message.getBytes());

    //Compute the message digest
    byte[] digest = md.digest();
    System.out.println(digest);

    //Converting the byte array in to HexString format
    StringBuffer hexString = new StringBuffer();

    for (int i = 0;i<digest.length;i++) {
      hexString.append(Integer.toHexString(0xFF & digest[i]));
    }
    System.out.println("Hex format : " + hexString.toString());
  }
}
```

Output

The above program generates the following output −

Enter the message
Hello how are you
[B@55f96302
Hex format: 2953d33828c395aebe8225236ba4e23fa75e6f13bd881b9056a3295cbd64d3

CONCLUSION :