

SAVITRIBAI PHULE PUNE UNIVERSITY

PES MODERN COLLEGE OF ENGINEERING



Computer Lab III

Author:
Devyani More

Supervisor:
Mrs.Aparna Junnarkar

A journal submitted in fullment of the requirements for the subject of "Computer Laboratory III"

June 2, 2016

PES's Modern College of Engineering
Department of Computer Engineering
Shivajinagar, Pune – 411005



CERTIFICATE

This is to certify that the Computer Laboratory 3 assignments submitted by

<u>Group Member</u>	<u>Roll No.</u>
Surabhi Gatagat	41008
Devyani More	41025
Kalpana Varat	41042
Janhavi Toshkhani	41078

Is a record of bonafide work carried out by them, in the partial fulfillment of the requirement for the award of Degree of B.E.(Computer Engineering) at **PES's Modern College of Engineering, Pune** under the **Savitribai Phule Pune University**. This work is done during year 2015-2016, under our guidance.

Place :

Date :

Internal Guide :
SUPERVISOR

External Examiner :
EXAMINER

Project Co-ordinator :
TEACHER

Head of the Department :
TEACHER

Contents

Certificate	ii
1 Binary Search Problem	1
1.1 Divide and Conquer Strategies	2
1.1.1 Divide and Conquer	2
1.2 Binary Search	2
1.3 Mathematical model	3
1.3.1 Input Set	3
1.3.2 Processing Set	3
1.3.3 Output Set	3
1.4 Modelio	4
1.5 Testing	5
1.5.1 Test Cases	5
1.6 Conclusion	5
2 Concurrent Quick Sort	7
2.1	8
2.2 2.1 Sorting Algorithm	8
2.2.1 2.1.1 Classification	8
2.3 Modelio	9
2.4 Test cases:	10
2.5 Conclusion	10
3 Booths Multiplication Algorithm	11
3.1 Introduction	11
3.1.1 Implementation	12
3.2 Advantages	14
3.3 Disadvantages	14
3.4 Test Case	15
3.5 Conclusion	15
4 Dining Philosopher's problem algorithm	17
4.1 Dining Philosopher's problem algorithm	18
4.2 Algorithm:	18
4.3 Mathematical model	19
4.3.1 INPUT:	19
4.3.2 PROCESSING:	19
4.3.3 OUTPUT:	19
4.4 Conclusion	19

5	A Mobile App for Calculator	21
5.1	Description	22
5.2	Platforms	22
5.3	Testing	23
5.3.1	Test Cases	23
5.4	Conclusion	23
6	Password data encryption	25
6.1	Theory	26
6.2	Purpose of encryption	26
6.3	Types of encryption	26
6.3.1	Symmetric key encryption	26
6.3.2	Public key encryption	26
6.4	Uses of encryption	27
6.5	Message verification	27
6.6	AES	28
6.7	Description of the cipher	28
6.8	Mathematical model	29
6.9	Test Cases	30
6.10	Conclusion	30
7	8-Queens Matrix	31
7.1	Eight-queens problem:	32
7.2	History	32
7.3	Solution Construction	32
7.4	Solutions	33
7.5	Mathematical model	34
7.5.1	Input	34
7.5.2	Processing	34
7.5.3	Output	34
7.6	Modelio	35
7.7	Testing	36
7.8	Conclusion	36
8	Calculator	37
8.1	Description	38
8.2	Platforms	38
8.3	Testing	39
8.3.1	Test Cases	39
8.4	Conclusion	39
9	8-Queens Matrix	41
9.1	Eight-Queens problem:	42
9.2	History	42
9.3	Solution Construction	42
9.4	Solutions	43
9.5	Mathematical model	44
9.6	Testing	45
9.7	Modelio	46
9.8	Conclusion	46

10 Signing based on SHA-1	47
10.1 Uninformed Search	48
10.1.1 DFS	48
10.1.2 BFS	48
Lowest-Cost-First Search	49
10.2 Mathematical Model	49
10.2.1 INPUT:	49
10.2.2 PROCESSING:	50
10.2.3 OUTPUT:	50
10.3 Testing	51
10.4 Testing	52
10.5 Conclusion	52
11 DSA signature	55
11.1 Description:	56
11.2 Explanation	56
11.3 Description:	56
11.4 How they work	57
11.4.1 For efficiency	58
11.4.2 For integrity	58
11.4.3 Notions of security	58
11.5 Applications of digital signatures	59
11.5.1 Authentication	59
11.5.2 Integrity	59
11.5.3 Non-repudiation	60
11.6 Mathematical Model	60
11.6.1 INPUT:	60
11.6.2 PROCESSING:	60
11.6.3 OUTPUT:	61
11.7 Conclusion	61
12 DSA signature	63
12.1 Description:	64
12.2 Explanation	64
12.3 Description:	64
12.4 How they work	65
12.4.1 For efficiency	66
12.4.2 For integrity	66
12.4.3 Notions of security	66
12.5 Applications of digital signatures	67
12.5.1 Authentication	67
12.5.2 Integrity	67
12.5.3 Non-repudiation	68
12.6 Miller-Rabin Test	68
12.7 Mathematical Model	69
12.7.1 INPUT:	69
12.7.2 PROCESSING:	69
12.7.3 OUTPUT:	69
12.8 Testing	70
12.9 Conclusion	70

13 Intrusion Detection System	71
13.1 Intrusion Detection System	72
13.2 Types of IDS	72
13.2.1 Network Intrusion Detection Systems (NIDS)	72
13.2.2 Host based Intrusion Detection Systems (HIDS)	73
13.3 Passive and reactive systems	73
13.4 Comparison with firewalls	73
13.5 Statistical anomaly and signature-based IDSes	74
13.5.1 Statistical anomaly-based IDS	74
13.5.2 Signature-based IDS	74
13.6 Limitations	74
13.7 Snort	75
13.7.1 Introduction	75
13.7.2 Uses	75
13.8 Installation of Snort	76
13.8.1 Pre-requisites	76
13.8.2 Installing Snort	77

List of Figures

1.1	Binary Search	5
5.1	Test case of Calculator	23
6.1	Test Cases of AES Encryption	29
6.2	Test Cases of AES Encryption	30
7.1	Chess Board : 8-Queens Problem	32
7.2	Test cases of 8-queens	36
8.1	Test case of Calculator	39
9.1	Chess Board : 8-Queens Problem	42
9.2	Flow graph :Test Cases of 8 queens Encryption	44
9.3	Flow graph :Test cases of 8-queens	45
10.1	Flow graph :Test cases of SHA1	51
10.2		51
12.1	Flow graph :Test cases of DSA	70

List of Tables

Assignment 1

Binary Search Problem

Title : Binary Search.

Problem Statement : 1. Write a program in Python/Java, using Divide and Conquer Strategies and Modelio software to design a function for Binary Search for an un-ordered data. Draw a USE-CASE diagram and mathematical model.

Strategies:

1. Binary Search Algorithm.
2. Divide and Conquer strategies.

Technologies Used :

- (a) Python

1.1 Divide and Conquer Strategies

The "Divide and conquer" technique involves in solving a particular computational problem by dividing it into smaller sub problems, solving the problem recursively and then combining the results of all the sub problems to produce the result for the original complex 'P'. A typical Divide and Conquer algorithm solves a problem using following three steps.

- (a) Divide: Break the given problem into sub problems of same type.
- (b) Conquer: Recursively solve these sub problems.
- (c) Combine: Appropriately combine the answers.

Following are some standard algorithms that are Divide and Conquer algorithms. Binary Search is a searching algorithm. In each step, the algorithm compares the input element x with the value of the middle element in array. If the values match, return the index of middle. Otherwise, if x is less than the middle element, then the algorithm recurs for left side of middle element, else recurs for right side of middle element.

1.1.1 Divide and Conquer

Let ' n ' represent the size of the original problem. Let $S(n)$ denote this problem. We solve the problem $S(n)$ by solving a collection of k sub problems- $S(n_1), S(n_2), \dots, S(n_k)$, Where $n_i < n$ for $i=1, 2, \dots, k$. Finally we merge the solutions to these problems. Suppose Let ' n ' be the maximum size of the problem. The number of smaller instances into which the input is divided is k . For an input of size ' n ' let $B(n)$ - No. of steps done to solve directly. $D(n)$ - No. of steps done by divide

For search algorithms, open list usually means the set of nodes yet to be explored and closed list the set of nodes that have already been explored.

1.2 Binary Search

Let a_i $1 < i < n$, be a list of element that are sorted in non decreasing order. We need to determine whether a given element ' x ' is present in the list. If ' x ' is present, then ' j ' is determined such that $a_j = x$. If x is not in the list, then j is to be set to zero. Solution: Every call to the binary search splits the partition into two in the middle according to the formula $Mid = [Low + high] / 2$ where low is the index of the first element of the partition and high is last element of the partition. Generally the ' n ' inputs are sorted in $A[1..n]$ array. The splitting process terminates when the size of the partition becomes '1' or required element is found. Organized the list in an order and first compare the required target key with the center element, and restrict our self to only first or second half of the list, depending on whether the key comes before or after the center one. In this way at each step we reduce the length by half, hence named as Binary Search.

Algorithm:

- (a) Read the 'n' numbers in the array.
- (b) Read the element to be searched.
- (c) $\text{Low}=0, \text{high}=n-1$ initialize the upper and lower limits.
- (d) Find the middle element $\text{Mid}=(\text{low}+\text{high})/2$ $y=a[\text{mid}]$
- (e) If $(X=Y)$ then print number found
- (f) If element is in upper half
- (g) $\text{High}=\text{Mid}-1$ goto 9 else $\text{low}=\text{mid}+1$.
- (h) If $(\text{high} \geq \text{low})$ then goto step 4
else print element is not in the list.
- (i) Stop.

1.3 Mathematical model

Let S be the set contains Input set ,processing set and output set. $S=I,F,O$

1.3.1 Input Set

Let I be the input set which contains sorted set of records and the record key to be searched. $I=I1, K$ $I1=R1, R2, R3, \dots, Rn$ Where $R1, R2, \dots, Rn$ are sorted records. $K=\text{Single key to be searched}$ $K=K1$ where $k1$ is the key to be searched.

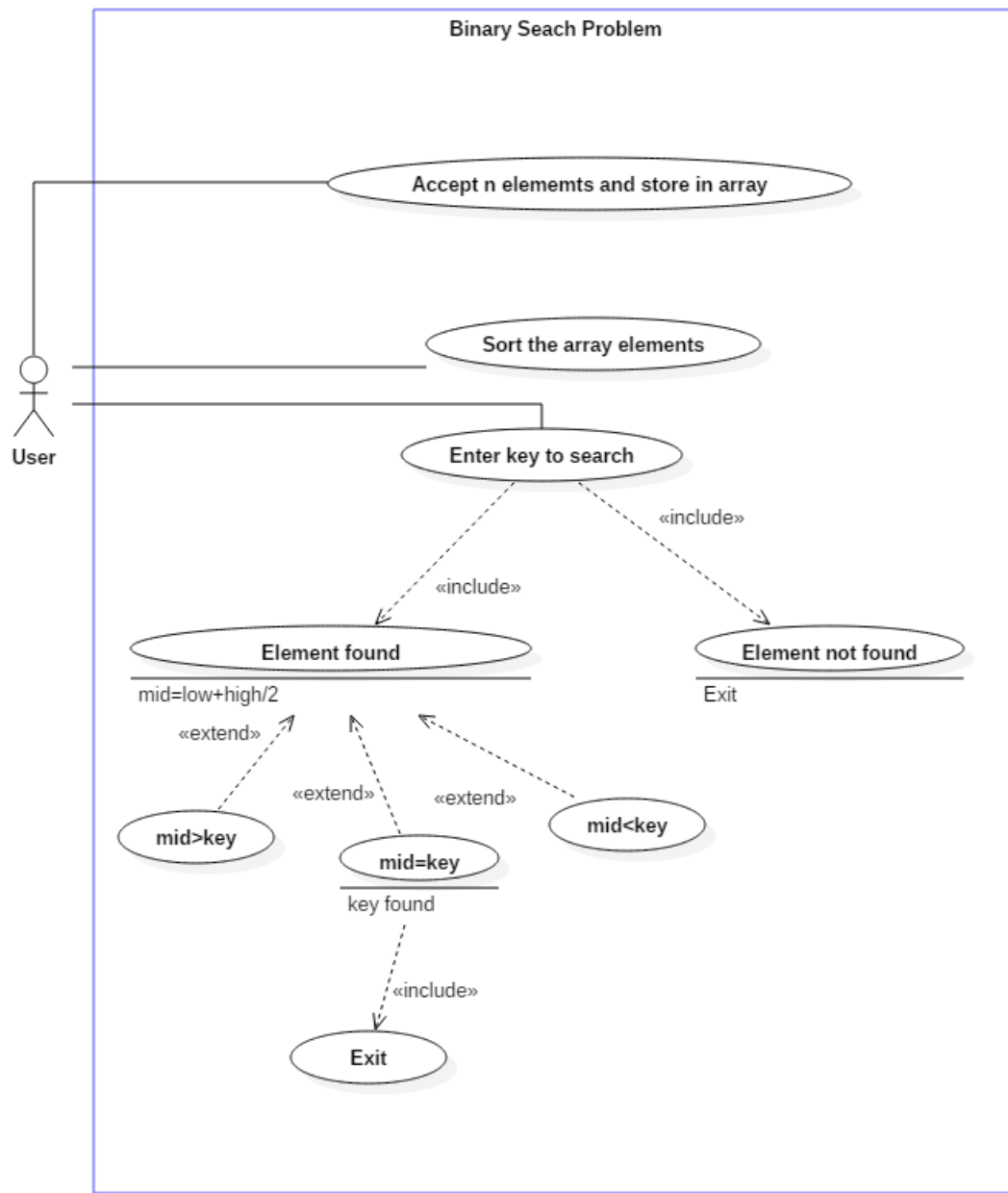
1.3.2 Processing Set

For processing our input and getting output, we have following functions:
 $F1=(\text{Accepting valid input})$ $O/P=(\text{Valid input nos})$
 $F2=(\text{Searching a key})$ $O/p=(\text{Key found})$

1.3.3 Output Set

In this set we will contain results as if we found the record in document or not. $O=\text{Key found, Key not found}$ And if key is found key is returned and if not msg that key is not found is returned.

1.4 Modelio



1.5 Testing

1.5.1 Test Cases

Tc id	Tc Description	Tc i/p	Expected output	Actual output	Result
P#01	Check whether the input is in integer	Integer Input from user	Valid input	Valid input	Pass
N#01	Check whether the special character input is accepted	Input from user	Invalid Input	Error message	Pass
BB#01	Check whether the input data is sorted	Sorted Input from user	Key found	Key found	Pass
WB#01	While first<= Last & not found midpoint=key	Input from user	Key found for single Element	Key found for single Element	Pass
WB#02	While first<=last & midpoint = key	Input from user	Key found on left array	Key found on left array	Pass
WB#03	While first<=last & not found midpoint>key	Input from user	Key found on right array	Key found on right array	Pass

FIGURE 1.1: Binary Search

1.6 Conclusion

Thus we have successfully implemented and learnt binary search algorithm using Sorted inputs.

Assignment 2

Concurrent Quick Sort

Title : Concurrent Quick Sort

Problem Statement : Implement Divide and Conquer Strategies to design an efficient class for Concurrent Quick Sort in Python/Java/C++, and the input data is stored using XML. Use Modelio. Draw a USE-CASE diagram and write mathematical model.]

Strategies:

- (a) Quick Sort Algorithm
- (b) Divide and Conquer Strategies

Technologies Used :

- i. Python

2.1

2.2 2.1 Sorting Algorithm

A sorting algorithm is an algorithm that puts elements of a list in a certain order. The most used orders are numerical order and lexicographical order. Efficient sorting is important for optimizing the use of other algorithms (such as search and merge algorithms) which require input data to be in sorted lists; it is also often useful for canonicalizing data and for producing human-readable output. More formally, the output must satisfy two conditions:

- (a) The output is in non-decreasing order (each element is no smaller than the previous element according to the desired total order);
- (b) The output is a permutation (reordering) of the input.

Further, the data is often taken to be in an array, which allows random access, rather than a list, which only allows sequential access, though often algorithms can be applied with suitable modification to either type of data.

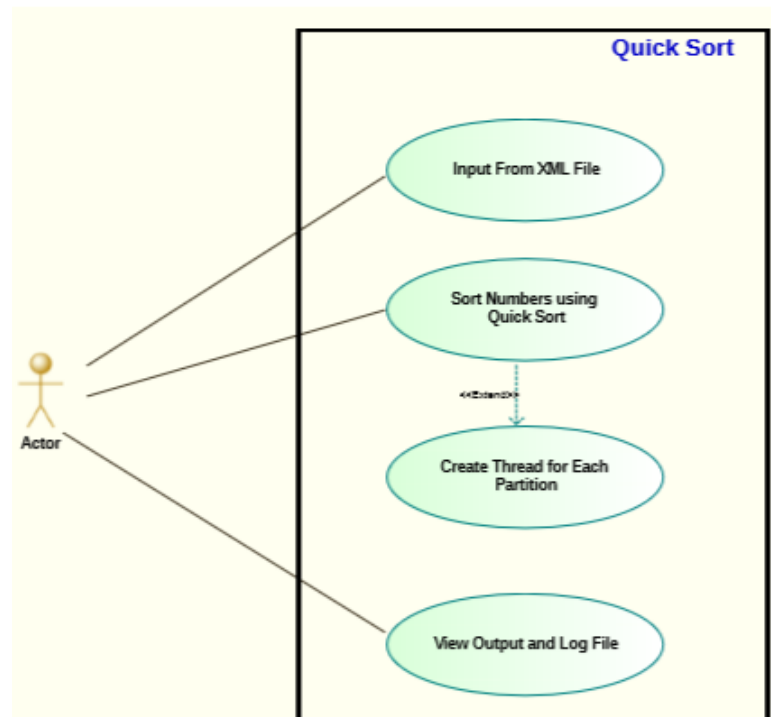
2.2.1 2.1.1 Classification

Sorting algorithms are often classified by:

- Computational complexity (worst, average and best behavior) of element comparisons in terms of the size of the list (n). For typical serial sorting algorithms good behavior is $O(n \log n)$, with parallel sort in $O(\log^2 n)$, and bad behavior is $O(n^2)$. Ideal behavior for a serial sort is $O(n)$, but this is not possible in the average case. Optimal parallel sorting is $O(\log n)$. Comparison-based sorting algorithms, which evaluate the elements of the list via an abstract key comparison operation, need at least $O(n \log n)$ comparisons for most inputs.
- Computational complexity of swaps (for “in place” algorithms).
- Memory usage (and use of other computer resources). In particular, some sorting algorithms are “in place”. Strictly, an in place sort needs only $O(1)$ memory beyond the items being sorted; sometimes $O(\log n)$ additional memory is considered “in place”.
- Recursion : Some algorithms are either recursive or non-recursive, while others may be both (e.g., merge sort).
- Stability : stable sorting algorithms maintain the relative order of records with equal keys (i.e., values).
- Whether or not they are a comparison sort. A comparison sort examines the data only by comparing two elements with a comparison operator.
- General method: insertion, exchange, selection, merging, etc. Exchange sorts include bubble sort and quick sort. Selection sorts include shaker sort and heap sort. Also whether the algorithm is serial or parallel. The remainder of this discussion almost exclusively concentrates upon serial algorithms and assumes serial operation.

- Adaptability: Whether or not the presortedness of the input affects the running time. Algorithms that take this into account are known to be adaptive.

2.3 Modelio



2.4 Test cases:

Sr.no	Tc Description	Tc i/p	Expected output	Actual output	Result
P#01	Check whether integer input are accepted	Integer input from XML file	Valid input	Valid input	Pass
N#01	Check whether character input are accepted	Character input from XML file	Shows INVALID	Error and Terminate	Pass
N#02	Check whether no content as input are accepted or not	Empty Input from Xml file	Invalid input	Invalid input	Pass
BB#01	Input Size=num Input limit=num	Valid size and limit of input	Sorted output	Sorted output	Pass
BB#02	Input Size=1 Input limit=num	Valid size, valid limit	Sorted output	Sorted output	Pass
WB#01	Perform quick sort	while(left<right)	Loop testing Till the time all the elements are not scanned the algorithm shouldn't end	Condition satisfied successfully	Pass
WB#02	Loop testing	if(high>low) setting bit to implement multithreading	Till the time partition are created thread should be assigned to each partition	Threads are created.	Pass

2.5 Conclusion

Thus using Divide and Conquer Strategies we can design an efficient class for Concurrent Quick Sort.

Assignment 3

Booths Multiplication Algorithm

Title : Booth's Multiplication Algorithm

Problem Statement : Implement a web Tool for Booth's multiplication algorithm which is used to multiply two numbers using HTML-5/Python/Java/C++. Use software design client-server architecture. Perform Positive and Negative testing. Use testing tool/Scrum-it/KADOS and Camel.

Strategies:

- (a) Booth's Algorithm
- (b) Web Technologies

Technologies Used :

- i. Python

3.1 Introduction

Booth's multiplication algorithm is a multiplication algorithm that multiplies two signed binary numbers in two's complement notation. Booth's algorithm is of interest in the study of computer architecture.

Algorithm Booth's algorithm examines adjacent pairs of bits of the N-bit multiplier Y in signed two's complement representation, including an implicit bit below the least significant bit, $y_{-1} = 0$. For each bit y_i , for i running from 0 to $N-1$, the bits y_i and y_{i-1} are considered. Where these two bits are equal, the product accumulator P is left unchanged. Where $y_i = 0$ and $y_{i-1} = 1$, the multiplicand times 2^i is added to P; and where $y_i = 1$ and $y_{i-1} = 0$, the multiplicand times 2^i is subtracted from P. The final value of P is the signed product.

The representations of the multiplicand and product are not specified; typically, these are both also in two's complement representation, like the multiplier, but any number system that supports addition and subtraction will work as well. As stated here, the order of

the steps is not determined. Typically, it proceeds from LSB to MSB, starting at $i = 0$; the multiplication by 2^i is then typically replaced by incremental shifting of the P accumulator to the right between steps; low bits can be shifted out, and subsequent additions and subtractions can then be done just on the highest N bits of $P[1]$. There are many variations and optimizations on these details.

The algorithm is often described as converting strings of 1's in the multiplier to a high-order $+1$ and a low-order -1 at the ends of the string. When a string runs through the MSB, there is no high-order $+1$, and the net effect is interpretation as a negative of the appropriate value.

3.1.1 Implementation

Implementation: Booth's algorithm can be implemented by repeatedly adding (with ordinary unsigned binary addition) one of two predetermined values A and S to a product P , then performing a rightward arithmetic shift on P . Let m and r be the multiplicand and multiplier, respectively; and let x and y represent the number of bits in m and r .

- (a) Determine the values of A and S , and the initial value of P . All of these numbers should have a length equal to $(x + y + 1)$.
- (b) A : Fill the most significant (leftmost) bits with the value of m . Fill the remaining $(y + 1)$ bits with zeros.
- (c) S : Fill the most significant bits with the value of (m) in two's complement notation. Fill the remaining $(y + 1)$ bits with zeros.
- (d) P : Fill the most significant x bits with zeros. To the right of this, append the value of r . Fill the least significant (rightmost) bit with a zero
- (a) Determine the two least significant (rightmost) bits of P .
 - i. If they are 01, find the value of $P + A$. Ignore any overflow.
 - ii. If they are 10, find the value of $P + S$. Ignore any overflow.
 - iii. If they are 00, do nothing. Use P directly in the next step.
 - iv. If they are 11, do nothing. Use P directly in the next step.
- (b) Repeat steps 2 and 3 until they have been done y times.
- (c) Drop the least significant (rightmost) bit from P . This is the product of m and r .

Example: Find $3 \times (4)$, with $m = 3$ and $r = 4$, and $x = 4$ and $y = 4$:

- (a) $m = 0011$, $-m = 1101$, $r = 1100$
- (b) $A = 0011\ 0000\ 0$
- (c) $S = 1101\ 0000\ 0$
- (d) $P = 0000\ 1100\ 0$
 - • Perform the loop four times:
- (a) $P = 0000\ 1100\ 0$. The last two bits are 00

$P = 0000\ 0110\ 0$. Arithmetic right shift.

- (b) $P = 0000\ 0110\ 0$. The last two bits are 00.
 $P = 0000\ 0011\ 0$. Arithmetic right shift.
- (c) $P = 0000\ 0011\ 0$. The last two bits are 10.
 $P = 1101\ 0011\ 0$. $P = P + S$.
 $P = 1110\ 1001\ 1$. Arithmetic right shift.
- (d) $P = 1110\ 1001\ 1$. The last two bits are 11.
 $P = 1111\ 0100\ 1$. Arithmetic right shift.
The product is 1111 0100, which is 12.

The above mentioned technique is inadequate when the multiplicand is the most negative number that can be represented (e.g. if the multiplicand has 4 bits then this value is 8). One possible correction to this problem is to add one more bit to the left of A, S and P. This then follows the implementation described above, with modifications in determining the bits of A and S; e.g., the value of m, originally assigned to the first x bits of A, will be assigned to the first x+1 bits of A. Below, we demonstrate the improved technique by multiplying 8 by 2 using 4 bits for the multiplicand and the multiplier:

$A = 1\ 1000\ 0000\ 0$

$S = 0\ 1000\ 0000\ 0$

$P = 0\ 0000\ 0010\ 0$

Perform the loop four times:

- (e) $P = 0\ 0000\ 0010\ 0$. The last two bits are 00.
 $P = 0\ 0000\ 0001\ 0$. Right shift.
- (f) $P = 0\ 0000\ 0001\ 0$. The last two bits are 10.
 $P = 0\ 1000\ 0001\ 0$. $P = P + S$.
 $P = 0\ 0100\ 0000\ 1$. Right shift.
- (g) $P = 0\ 0100\ 0000\ 1$. The last two bits are 01.
 $P = 1\ 1100\ 0000\ 1$. $P = P + A$.
 $P = 1\ 1110\ 0000\ 0$. Right shift.
- (h) $P = 1\ 1110\ 0000\ 0$. The last two bits are 00.
 $P = 1\ 1111\ 0000\ 0$. Right shift.
- (i) The product is 11110000 (after discarding the first and the last bit) which is 16. How it works: Consider a positive multiplier consisting of a block of 1s surrounded by 0s. For example, 00111110. The product is given by:

where M is the multiplicand. The number of operations can be reduced to two by rewriting the same as

In fact, it can be shown that any sequence of 1's in a binary number can be broken into the difference of two binary numbers:

Hence, we can actually replace the multiplication by the string of ones in the original number by simpler operations, adding the multiplier, shifting the partial product thus formed by appropriate places, and then finally subtracting the multiplier. It is making use of the fact that we do not have to do anything but shift while we are dealing with 0s in a binary multiplier, and is similar to using the mathematical property that $99 = 100 - 1$ while multiplying by 99. This scheme can be extended to any number of blocks of 1s in a multiplier (including the case of a single 1 in a block). Thus,

Booth's algorithm follows this scheme by performing an addition when it encounters the first digit of a block of ones (0 1) and a subtraction when it encounters the end of the block (1 0). This works for a negative multiplier as well. When the ones in a multiplier are grouped into long blocks, Booth's algorithm performs fewer additions and subtractions than the normal multiplication algorithm.

3.2 Advantages

It has following advantages

- i. Resource sharing Sharing of hardware and software resources.
- ii. Openness Flexibility of using hardware and software of different vendors.
- iii. Concurrency Concurrent processing to enhance performance.
- iv. Scalability Increased throughput by adding new resources.
- v. Fault tolerance The ability to continue in operation after a fault has occurred.

3.3 Disadvantages

Its disadvantages are

- i. Complexity They are more complex than centralized systems.
- ii. Security More susceptible to external attack.
- iii. Manageability More effort required for system management.
- iv. Unpredictability Unpredictable responses depending on the system organization and network load.

3.4 Test Case

Tc id	Tc Description	Tc i/p	Expected output	Actual output	Result
RT#01	Check whether Server Acknowledges Client.	Client request to server	Client receives Acknowledgement from server	Client receives Acknowledgement From server	Pass
RT#01	Check whether application is compatible on various browsers like Chrome, Firefox and Explorer.	Browsing application on various browsers.	Compatible with all browsers like Chrome , Firefox and explorer.	Compatible with all browsers like like Chrome, Firefox and Explorer.	Pass
RT#01	Check whether GUI is Displayed every time application is run.	Application Output	Proper GUI obtained	Proper GUI obtained	Pass
P#01	Check whether the input is an integer	Integer Input from user	Valid input	Valid input	Pass
N#01	Check whether the special character input is accepted	Input from user in special character	Invalid input	Error message	Pass
N#01	Check whether more than two inputs are accepted	More than 2 input from user	invalid input	Error message	pass
WB#01	Check for positive input number	Positive Integer Input from user	No operation performed on positive input	No operation performed on positive input	Pass
WB#01	Check for negative input number	Negative Integer Input from user	2's complement performed on negative input	2's complement performed on negative input.	Pass

3.5 Conclusion

Thus a Web Tool for Booth's multiplication algorithm is used to multiply two numbers located in distributed environment.

Assignment 4

Dining Philosopher's problem algorithm

Title : Dining Philosopher's problem algorithm

Problem Statement : Write a program in Python/ C++/ Java for Dining Philosopher's problem algorithm which is used to design a software that uses shared memory between neighboring processes to consume the data. The Data is stored in MongoDB (NoSQL). Design using Client-Server architecture. Use testing tool/Scrum-it/KADOS, NoSQLUnit and Camel.

Strategies:

- i. Dining Philosopher's problem algorithm

Technologies Used :

- i. MongoDB

4.1 Dining Philosopher's problem algorithm

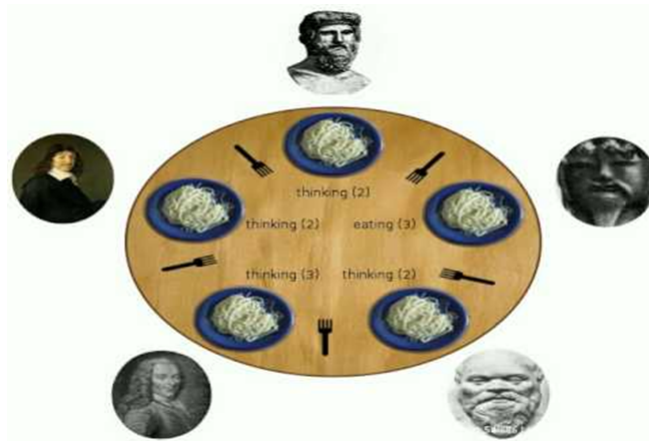
Dining Philosopher's problem, in computer science and logic, is an algorithmic process of solving equations between symbolic expressions.

The dining philosophers problem is an example problem often used in concurrent algorithm design to illustrate synchronization issues and techniques for resolving them. "Five philosophers are sitting around a round table. Each has in front of him a bowl of rice. Between each philosopher and his neighbour is a chopstick; Five chopsticks in all. Each philosopher thinks for a while, then gets hungry and wants to eat, he has to get two chopsticks. Once he possesses two chopsticks, he will eat for a while, then put both chopsticks back down on each side."

A solution to this problem is an algorithm for each of the philosophers to follow that will ensure that- 1. There will never be a situation of deadlock i.e. no philosopher gets to eat. 2. There will never be a situation of resource starvation. That is, a situation in which philosopher wants to eat but never does so. the situation of deadlock is as follows: Suppose that all philosophers become hungry at the same time and take their left fork, and then if all reach for right fork no one can move.

4.2 Algorithm:

- i. Move from thinking stage to hungry.
- ii. When hungry, randomly choose to try and pick up the left or right fork.
- iii. Wait until the fork is down and then pick it up.
- iv. If the other fork is free, pick it up; otherwise put the original fork down and return to step 1.
- v. As both the forks are in possession, eat.
- vi. After finished eating put both forks down and return to thinking stage (Start thinking!!).



4.3 Mathematical model

4.3.1 INPUT:

Table = $T = P, C, B$ | where P is philosophers, C is chopsticks, B is bowl

4.3.2 PROCESSING:

Philosopher = TH, LC, E, ULC | where Th, E, P, TH = thinking time for a particular philosopher LC = lock chopstick = CR, CL | where C, PI and i is the philosopher picking the chopsticks. A philosopher can pick only immediate left and immediate right chopsticks. E = eating time for a particular philosopher ULC = unlock chopstick = CR, CL | CL, CR, PI, S = starvation = if $((E \text{ and } TH) \text{ of a } P) < \text{rest of } P$, then this philosopher will think fast and get hungry fast $LC(P) < \text{rest of } P$. Other philosophers will starve. If E, TH, P is equivalent then $S(P) = 0$

4.3.3 OUTPUT:

Each philosopher gets enough time to eat and think

4.4 Conclusion

Thus Dining Philosopher's problem algorithm is used to design a software that uses shared memory between neighboring processes to consume the data.

Assignment 5

A Mobile App for Calculator

Title : A Mobile App for Calculator

Problem Statement : Implement A Mobile App for Calculator having Trigonometry functionality which is to be designed using HTML-5/Python/Java/C++. The data storage uses text files. Use testing tool/ Scrum-it. Perform Positive and Negative testing.

Strategies :

- i. Scientific Calculator Functions
- ii. Java

Technologies Used :

- i. HTML-5
- ii. Js
- iii. NetBeans

5.1 Description

Advanced Trigonometry Calculator is a rock-solid calculator allowing you perform advanced complex math calculations. Enter your complex math expression on its integrity and in the final press “Enter” button, after some instants the solution for your expression will be displayed. Anyone can use this calculator since the syntax used is very similar with scientific handheld calculators, e.g. TI 84-Plus.

A software calculator is a calculator that has been implemented as a computer program, rather than as a physical hardware device. They are among the simpler interactive software tools, and, as such, they:

- Provide operations for the user to select one at a time.
 - Can be used to perform any process that consists of a sequence of steps each of which applies one of these operations.
 - Have no purpose other than these processes, because the operations are the sole, or at least the primary, features of the calculator, rather than being secondary features that support other functionality that is not normally known simply as calculation.
- As a calculator, rather than a computer, they usually: Have a small set of relatively simple operations. Perform short processes that are not compute intensive. Do not accept large amounts of input data or produce many results.

5.2 Platforms

Software calculators are available for many different platforms, and they can be:

- A program for, or included with an operating system.
- A program implemented as server or client-side scripting (such as JavaScript) within a web page.
- Embedded in a calculator watch.
- Also complex software may have calculator-like dialogs, sometimes with the full calculator functionality, to enter data into the system.

5.3 Testing

5.3.1 Test Cases

Tc id	Tc Description	Tc i/p	Expected output	Actual output	Result
P#01	Check whether the input is an integer	Integer Input from user	Valid input	Valid input	Pass
P#01	Check whether the Float input are accepted.	Float Input from user	Valid input	Valid input	Pass
N#01	Check whether more than one Special character are accepted	Input with more than one special character	Invalid input	Invalid input	Pass
N#01	Check whether output for one input and a operation is obtained	Input from user	Invalid Input	Invalid input	Pass
N#01	Check whether more than two inputs are accepted	More than 2 Input from user	Invalid input	Invalid input	pass
BB#01	Check whether two inputs are accepted	2 Input from user	valid input	valid input	pass
WB#01	Check whether multiplication of two inputs is performed	2 Input from user	Valid output	Valid output	Pass
WB#01	Check whether sin of a input is performed	Input from user	Valid output	Valid output	Pass

FIGURE 5.1: Test case of Calculator

5.4 Conclusion

Thus a Mobile App for Calculator having Trigonometry functionality is designed and tested.

Assignment 6

Password data encryption

Title : Password data encryption

Problem Statement : Write a program in python/ Java/ C++ to implement password data encryption. Use encryption method overloading (any two methods).

Strategies:

(a) AES

Technologies Used :

(a) Java

6.1 Theory

In cryptography, encryption is the process of encoding messages or information in such a way that only authorized parties can read it. Encryption does not of itself prevent interception, but denies the message content to the interceptor. In an encryption scheme, the intended communication information or message, referred to as plaintext, is encrypted using an encryption algorithm, generating ciphertext that can only be read if decrypted. For technical reasons, an encryption scheme usually uses a pseudo-random encryption key generated by an algorithm. It is in principle possible to decrypt the message without possessing the key, but, for a well-designed encryption scheme, large computational resources and skill are required. An authorized recipient can easily decrypt the message with the key provided by the originator to recipients, but not to unauthorized interceptors.

6.2 Purpose of encryption

The purpose of encryption is to ensure that only somebody who is authorized to access data (e.g. a text message or a file), will be able to read it, using the decryption key. Somebody who is not authorized can be excluded, because he or she does not have the required key, without which it is impossible to read the encrypted information.

6.3 Types of encryption

6.3.1 Symmetric key encryption

In symmetric-key schemes,[1] the encryption and decryption keys are the same. Communicating parties must have the same key before they can achieve secure communication.

6.3.2 Public key encryption

Illustration of how encryption is used within servers Public key encryption. In public-key encryption schemes, the encryption key is published for anyone to use and encrypt messages. However, only the receiving party has access to the decryption key that enables messages to be read.[2] Public-key encryption was first described in a secret document in 1973;[3] before then all encryption schemes were symmetric-key (also called private-key).[4]:478 A publicly available public key encryption application called Pretty Good Privacy (PGP) was written in 1991 by Phil Zimmermann, and distributed free of charge with source code; it was purchased by Symantec in 2010 and is regularly updated.[5]

6.4 Uses of encryption

Encryption has long been used by military and governments to facilitate secret communication. It is now commonly used in protecting information within many kinds of civilian systems. For example, the Computer Security Institute reported that in 2007, 71 percent of companies surveyed utilized encryption for some of their data in transit, and 53 percent utilized encryption for some of their data in storage.[6] Encryption can be used to protect data "at rest", such as information stored on computers and storage devices (e.g. USB flash drives). In recent years there have been numerous reports of confidential data such as customers' personal records being exposed through loss or theft of laptops or backup drives. Encrypting such files at rest helps protect them should physical security measures fail. Digital rights management systems, which prevent unauthorized use or reproduction of copyrighted material and protect software against reverse engineering (see also copy protection), is another somewhat different example of using encryption on data at rest.[7] Encryption is also used to protect data in transit, for example data being transferred via networks (e.g. the Internet, e-commerce), mobile telephones, wireless microphones, wireless intercom systems, Bluetooth devices and bank automatic teller machines. There have been numerous reports of data in transit being intercepted in recent years.[8] Data should also be encrypted when transmitted across networks in order to protect against eavesdropping of network traffic by unauthorized users.[9]

6.5 Message verification

Encryption, by itself, can protect the confidentiality of messages, but other techniques are still needed to protect the integrity and authenticity of a message; for example, verification of a message authentication code (MAC) or a digital signature. Standards for cryptographic software and hardware to perform encryption are widely available, but successfully using encryption to ensure security may be a challenging problem. A single error in system design or execution can allow successful attacks. Sometimes an adversary can obtain unencrypted information without directly undoing the encryption. See, e.g., traffic analysis, TEMPEST, or Trojan horse.[10] Digital signature and encryption must be applied to the ciphertext when it is created (typically on the same device used to compose the message) to avoid tampering; otherwise any node between the sender and the encryption agent could potentially tamper with it. Encrypting at the time of creation is only secure if the encryption device itself has not been tampered with.

6.6 AES

The Advanced Encryption Standard (AES), also known as Rijndael[4][5] (its original name), is a specification for the encryption of electronic data established by the U.S. National Institute of Standards and Technology (NIST) in 2001.[6] AES is based on the Rijndael cipher[5] developed by two Belgian cryptographers, Joan Daemen and Vincent Rijmen, who submitted a proposal to NIST during the AES selection process.[7] Rijndael is a family of ciphers with different key and block sizes. For AES, NIST selected three members of the Rijndael family, each with a block size of 128 bits, but three different key lengths: 128, 192 and 256 bits. The Advanced Encryption Standard (AES) is defined in each of:

- FIPS PUB 197: Advanced Encryption Standard (AES)[6]
- ISO/IEC 18033-3: Information technology — Security techniques — Encryption algorithms — Part 3: Block ciphers

6.7 Description of the cipher

AES is based on a design principle known as a substitution-permutation network, combination of both substitution and permutation, and is fast in both software and hardware.[10] Unlike its predecessor DES, AES does not use a Feistel network. AES is a variant of Rijndael which has a fixed block size of 128 bits, and a key size of 128, 192, or 256 bits. By contrast, the Rijndael specification per se is specified with block and key sizes that may be any multiple of 32 bits, both with a minimum of 128 and a maximum of 256 bits. AES operates on a 44 column-major order matrix of bytes, termed the state, although some versions of Rijndael have a larger block size and have additional columns in the state. Most AES calculations are done in a special finite field.

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

- 10 cycles of repetition for 128-bit keys.
- 12 cycles of repetition for 192-bit keys.
- 14 cycles of repetition for 256-bit keys. Each round consists of several processing steps, each containing four similar but different stages, including one that depends on the encryption key itself. A set of reverse rounds are applied to transform ciphertext back into the original plaintext using the same encryption key.

6.8 Mathematical model

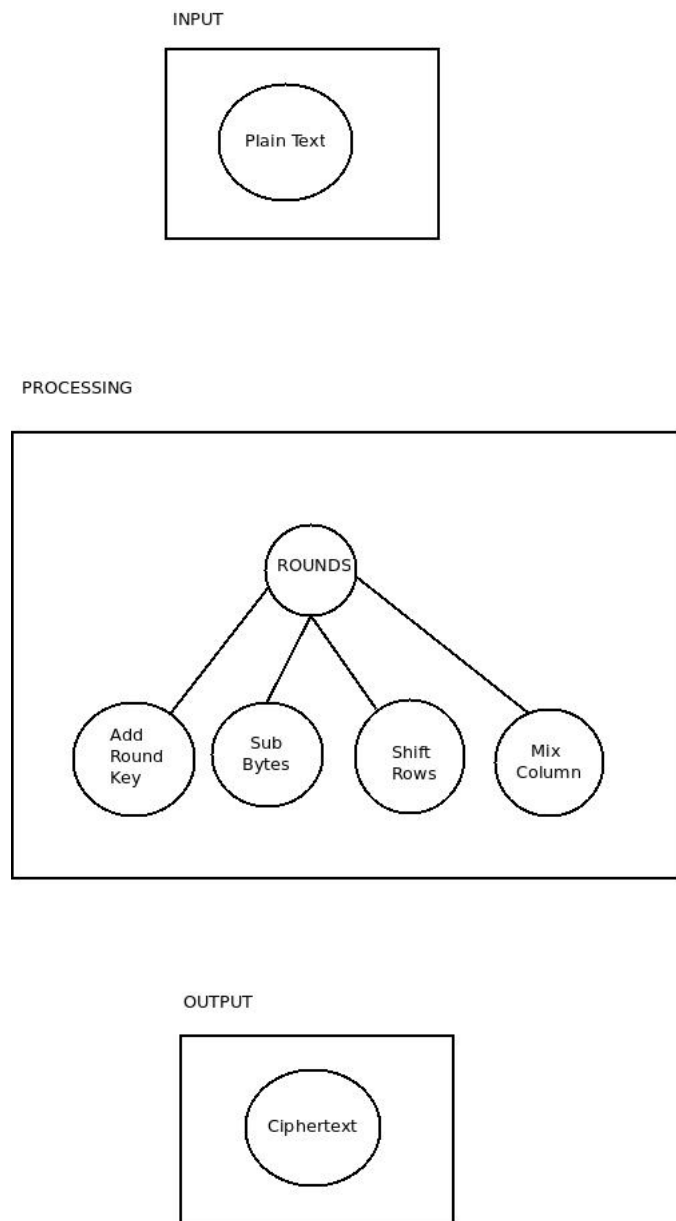


FIGURE 6.1: Test Cases of AES Encryption

6.9 Test Cases

Sr.no	Tc Description	Tc i/p	Expected output	Actual output	Result
P#01	Check whether string plaintext are accepted	String plaintext	Valid input	Valid input	Pass
P#02	Check whether the input generates cipher text.	String plaintext	Cipher text	Cipher text	Pass
P#03	Check whether the cipher text generates plaintext	Cipher text	Plaintext	Plaintext	Pass
N#01	Check whether input of regional languages are accepted	Regional languages	Invalid input	Invalid input	Pass
BB#01	Check whether 128 Bit key is accepted.	128 bit key	Valid key input	Valid key Input	pass
WB#01	Check for Roundkey(i) where $i < 10$	Roundkey(i)	Cipher text	Cipher text	Pass

FIGURE 6.2: Test Cases of AES Encryption

6.10 Conclusion

Thus in password data encryption we understood encryption of plaintext and decryption of encrypted message using AES.

Assignment 7

8-Queens Matrix

Title : 8-Queens Matrix

Problem Statement : 1. Write a program in Python/ Java for 8-Queens Matrix which is Stored using JSON/XML having first Queen placed. Use back-tracking strategy to place remaining Queens. Generate final 8-queen's Matrix.

Pre-requisites :

- (a) 8-queen problem
- (b) Java

7.1 Eight-queens problem:

Eight-queens Problem The eight queens puzzle is the problem of placing eight chess queens on an 8x8 chessboard so that no two queens threaten each other. Thus, a solution requires that no two queens share the same row, column, or diagonal. The eight queens puzzle is an example of the more general n-queens problem of placing n queens on an nxn chessboard, where solutions exist for all natural numbers n with the exception of n=2 and n=3.

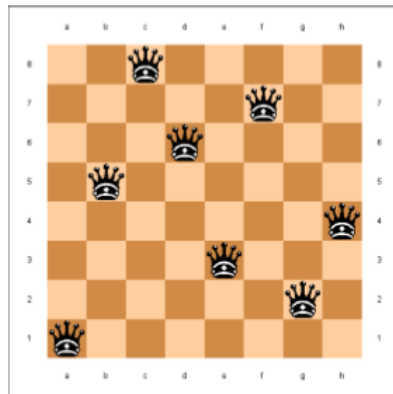


FIGURE 7.1: Chess Board : 8-Queens Problem

7.2 History

Chess composer Max Bezzel published the eight queens puzzle in 1848. Franz Nauck published the first solutions in 1850. Nauck also extended the puzzle to the n-queens problem, with n queens on a chessboard of nxn squares. Since then, many mathematicians, including Carl Friedrich Gauss, have worked on both the eight queens puzzle and its generalized n-queens version. In 1874, S. Gunther proposed a method using determinants to find solutions. J.W.L. Glaisher refined Gunther's approach.

In 1972 Edsger Dijkstra used this problem to illustrate the power of what he called structured programming. He published a highly detailed description of a depth-first backtracking algorithm.

7.3 Solution Construction

The problem can be quite computationally expensive as there are 4,426,165,368 (i.e., $64C8$) possible arrangements of eight queens on an 8x8 board, but only 92 solutions. It is possible to use shortcuts that reduce computational requirements or rules of thumb that avoid brute-force computational techniques.

For example, just by applying a simple rule that constrains each queen to a single column (or row), though still considered brute force, it is possible to reduce the number of possibilities to just 16,777,216 (that is, 8^8) possible combinations. Generating permutations further reduces the possibilities to just 40,320 (that is, $8!$), which are then checked for diagonal attacks. Martin Richards published a program to count solutions to the n -queens problem using bitwise operations.

7.4 Solutions

The eight queens puzzle has 92 distinct solutions. If solutions that differ only by symmetry operations (rotations and reflections) of the board are counted as one, the puzzle has 12 fundamental solutions. A fundamental solution usually has eight variants (including its original form) obtained by rotating 90, 180, or 270 and then reflecting each of the four rotational variants in a mirror in a fixed position. However, should a solution be equivalent to its own 90 rotation (as happens to one solution with 8 queens on an 8x8 board) that fundamental solution will have only two variants (itself and its reflection). Should a solution be equivalent to its own 180 rotation (but not to its 90 rotation) it will have four variants (itself, its reflection, its 90 rotation and the reflection of that). It is not possible for a solution to be equivalent to its own reflection (except at $n=1$) because that would require two queens in the same row. (For n -queen problem's solution to be equivalent to its own mirror-image solution, the solution needs to be symmetrical by the center of the board either horizontally or vertically. Then, two queens would be facing each other, making it not a solution.) Of the 12 fundamental solutions to the problem with eight queens on an 8x8 board, exactly one is equal to its own 180 rotation, and none are equal to their 90 rotation, thus the number of distinct solutions is $11 \times 8 + 1 \times 4 = 92$ (where the 8 is derived from four 90 rotational positions and their reflections, and the 4 is derived from two 180 rotational positions and their reflections).

7.5 Mathematical model

8 queens

7.5.1 Input

Let A be a set such that A=set of xml file having first queens chess board position

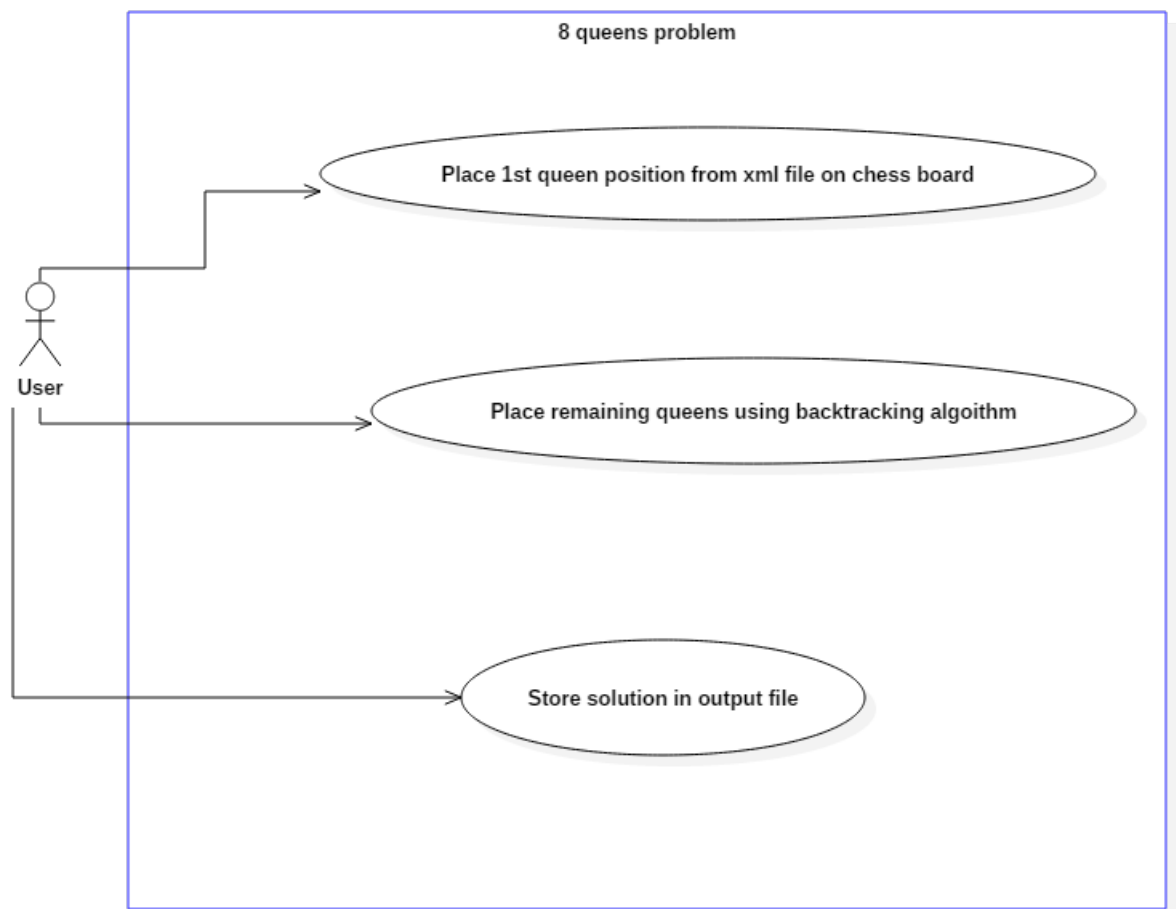
7.5.2 Processing

Let B=set of module to match the position of queen with row and column C=set of module to backtrack the position of queen B compares the current position with the row and column conflict of the other queen. If conflict then backtrack else place the queen on the position C will only be executed if B gives a conflict condition of the queen position. Thus Processing $P=B \cup C$

7.5.3 Output

Let O=set of output file having all 8 queens placed on board without any position conflict

7.6 Modelio



7.7 Testing

Tc id	Tc Description	Tc i/p	Expected output	Actual output	Result
P#01	User input(Binary)	Input of single Queen position.	Solution should be found	Solution Found	Pass
N#01	Check whether the inputs(more than two position for Queen) are accepted	Input of Queen positions.	Invalid input	Invalid input	Pass
BB#01	XML file input	Character input reading from XML file	Shows INVALID	Error and Terminate	Pass
WB#02	XML file input	Integer input reading from XML file where value of i is greater than 8	Shows INVALID	Error and Terminate	Pass

FIGURE 7.2: Test cases of 8-queens

7.8 Conclusion

Thus 8-Queens Matrix is Stored using JSON/XML.

Assignment 8

Calculator

Title : Mobile App for Calculator

Problem Statement : A mobile application needs to be designed for using a Calculator (+, - , *, / , Sin, Cos, sq-root) with Memory Save/Recall using Extended precision floating point number format. Give the Required modeling, Design and Positive-Negative test cases.

Strategies :

- (a) Scientific Calculator Functions
- (b) Java

Technologies Used :

- (a) HTML-5
- (b) Js
- (c) NetBeans

8.1 Description

Advanced Trigonometry Calculator is a rock-solid calculator allowing you perform advanced complex math calculations. Enter your complex math expression on its integrity and in the final press “Enter” button, after some instants the solution for your expression will be displayed. Anyone can use this calculator since the syntax used is very similar with scientific handheld calculators, e.g. TI 84-Plus.

A software calculator is a calculator that has been implemented as a computer program, rather than as a physical hardware device. They are among the simpler interactive software tools, and, as such, they:

- Provide operations for the user to select one at a time.
- Can be used to perform any process that consists of a sequence of steps each of which applies one of these operations.
- Have no purpose other than these processes, because the operations are the sole, or at least the primary, features of the calculator, rather than being secondary features that support other functionality that is not normally known simply as calculation.

As a calculator, rather than a computer, they usually: Have a small set of relatively simple operations. Perform short processes that are not compute intensive. Do not accept large amounts of input data or produce many results.

8.2 Platforms

Software calculators are available for many different platforms, and they can be:

- A program for, or included with an operating system.
- A program implemented as server or client-side scripting (such as JavaScript) within a web page.
- Embedded in a calculator watch.
- Also complex software may have calculator-like dialogs, sometimes with the full calculator functionality, to enter data into the system.

8.3 Testing

8.3.1 Test Cases

Tc id	Tc Description	Tc i/p	Expected output	Actual output	Result
P#01	Check whether the input is an integer	Integer Input from user	Valid input	Valid input	Pass
P#01	Check whether the Float input are accepted.	Float Input from user	Valid input	Valid input	Pass
N#01	Check whether more than one Special character are accepted	Input with more than one special character	Invalid input	Invalid input	Pass
N#01	Check whether output for one input and a operation is obtained	Input from user	Invalid Input	Invalid input	Pass
N#01	Check whether more than two inputs are accepted	More than 2 Input from user	Invalid input	Invalid input	pass
BB#01	Check whether two inputs are accepted	2 Input from user	valid input	valid input	pass
WB#01	Check whether multiplication of two inputs is performed	2 Input from user	Valid output	Valid output	Pass
WB#01	Check whether sin of a input is performed	Input from user	Valid output	Valid output	Pass

FIGURE 8.1: Test case of Calculator

8.4 Conclusion

Thus a web application using Scala/ Python/ Java /HTML5 to check the plagiarism in the given text paragraph written/ copied in the text box.

Assignment 9

8-Queens Matrix

Title : 8-Queens Matrix

Problem Statement : Write a program in Python/ Java for 8-Queens Matrix which is Stored using JSON/XML having first Queen placed, use back-tracking to place remaining Queens to generate final 8-queen's Matrix. Use suitable testing methods.

Pre-requisites :

- (a) 8-queen problem

Technologies Used :

- (a) 8-queen problem
- (b) Java

9.1 Eight-Queens problem:

Eight-Queens Problem The eight queens puzzle is the problem of placing eight chess queens on an 8x8 chessboard so that no two queens threaten each other. Thus, a solution requires that no two queens share the same row, column, or diagonal. The eight queens puzzle is an example of the more general n-queens problem of placing n queens on an nxn chessboard, where solutions exist for all natural numbers n with the exception of n=2 and n=3.

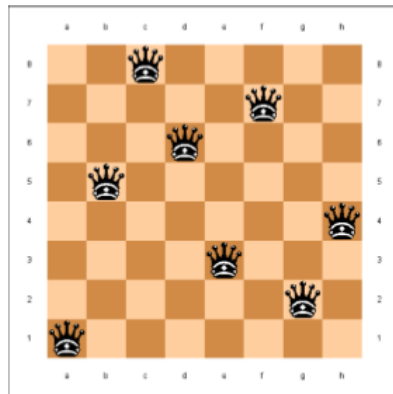


FIGURE 9.1: Chess Board : 8-Queens Problem

9.2 History

Chess composer Max Bezzel published the eight queens puzzle in 1848. Franz Nauck published the first solutions in 1850. Nauck also extended the puzzle to the n-queens problem, with n queens on a chessboard of nxn squares. Since then, many mathematicians, including Carl Friedrich Gauss, have worked on both the eight queens puzzle and its generalized n-queens version. In 1874, S. Gunther proposed a method using determinants to find solutions. J.W.L. Glaisher refined Gunther's approach.

In 1972 Edsger Dijkstra used this problem to illustrate the power of what he called structured programming. He published a highly detailed description of a depth-first backtracking algorithm.

9.3 Solution Construction

The problem can be quite computationally expensive as there are 4,426,165,368 (i.e., $8!$) possible arrangements of eight queens on an 8x8 board, but only 92 solutions. It is possible to use shortcuts that reduce computational requirements or rules of thumb that avoid brute-force computational techniques. For

example, just by applying a simple rule that constrains each queen to a single column (or row), though still considered brute force, it is possible to reduce the number of possibilities to just 16,777,216 (that is, 8^8) possible combinations. Generating permutations further reduces the possibilities to just 40,320 (that is, $8!$), which are then checked for diagonal attacks. Martin Richards published a program to count solutions to the n-queens problem using bitwise operations.

9.4 Solutions

The eight queens puzzle has 92 distinct solutions. If solutions that differ only by symmetry operations (rotations and reflections) of the board are counted as one, the puzzle has 12 fundamental solutions. A fundamental solution usually has eight variants (including its original form) obtained by rotating 90, 180, or 270 and then reflecting each of the four rotational variants in a mirror in a fixed position. However, should a solution be equivalent to its own 90 rotation (as happens to one solution with 8 queens on a 5x5 board) that fundamental solution will have only two variants (itself and its reflection).

Should a solution be equivalent to its own 180 rotation (but not to its 90 rotation) it will have four variants (itself, its reflection, its 90 rotation and the reflection of that). It is not possible for a solution to be equivalent to its own reflection (except at $n=1$) because that would require two queens to be facing each other. (For n-queen problem's solution to be equivalent to its own mirror-image solution, the solution needs to be symmetrical by the center of the board either horizontally or vertically. Then, two queens would be facing each other, making it not a solution.) Of the 12 fundamental solutions to the problem with eight queens on an 8x8 board, exactly one is equal to its own 180 rotation, and none are equal to their 90 rotation, thus the number of distinct solutions is $118 + 14 = 92$ (where the 8 is derived from four 90 rotational positions and their reflections, and the 4 is derived from two 180 rotational positions and their reflections).

9.5 Mathematical model

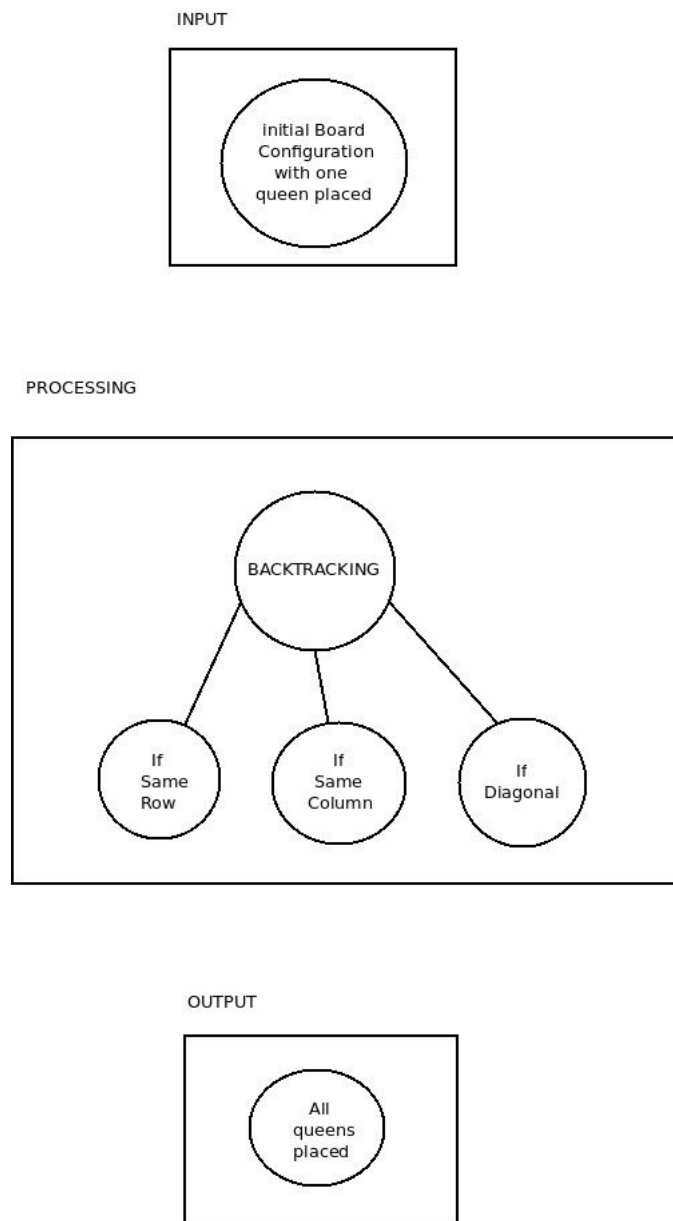


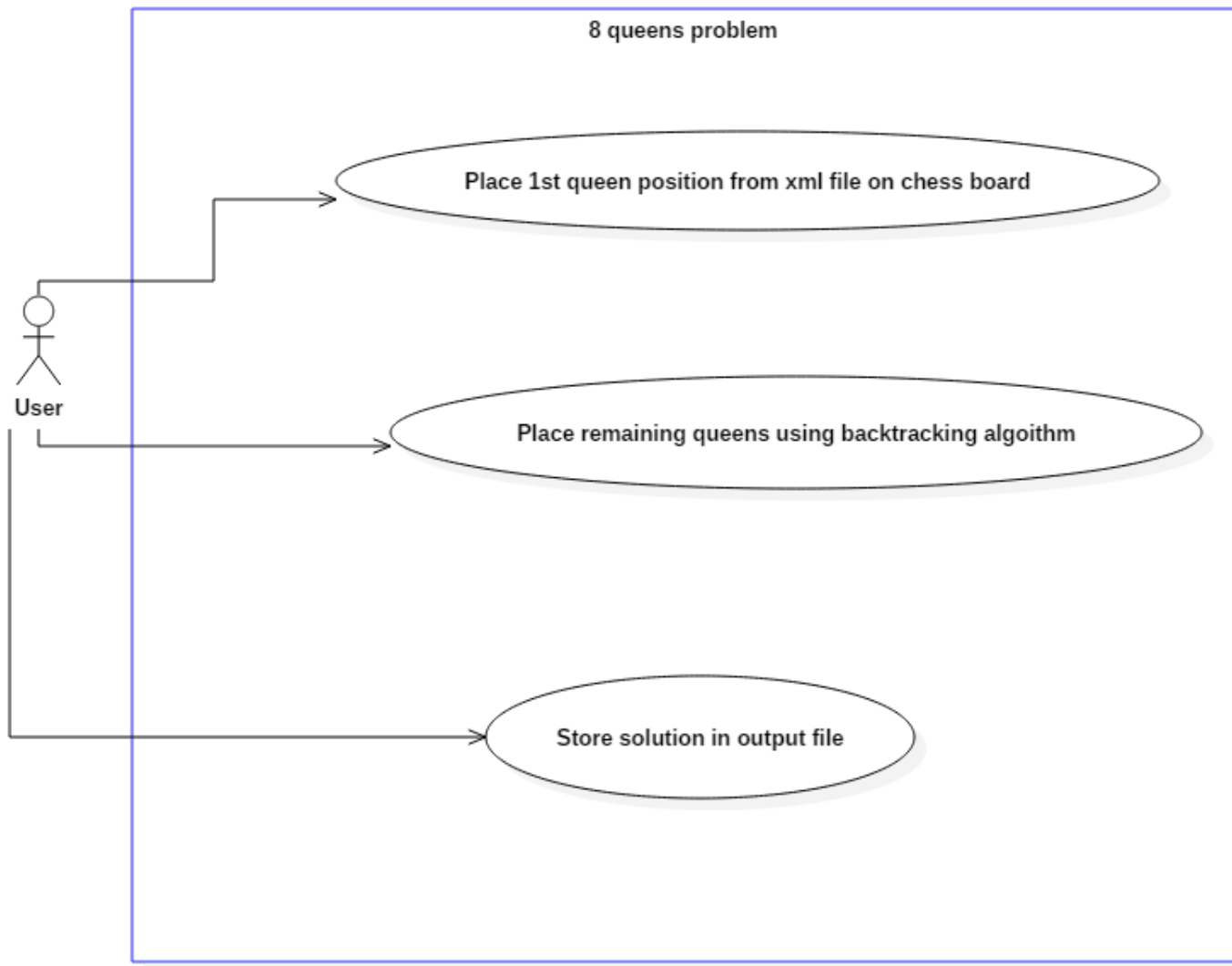
FIGURE 9.2: Flow graph :Test Cases of 8 queens Encryption

9.6 Testing

Tc id	Tc Description	Tc i/p	Expected output	Actual output	Result
P#01	User input(Binary)	Input of single Queen position.	Solution should be found	Solution Found	Pass
N#01	Check whether the inputs(more than two position for Queen) are accepted	Input of Queen positions.	Invalid input	Invalid input	Pass
BB#01	XML file input	Character input reading from XML file	Shows INVALID	Error and Terminate	Pass
WB#02	XML file input	Integer input reading from XML file where value of i is greater than 8	Shows INVALID	Error and Terminate	Pass

FIGURE 9.3: Flow graph :Test cases of 8-queens

9.7 Modelio



9.8 Conclusion

Thus 8-Queens Matrix is Stored using JSON/XML.

Assignment 10

Signing based on SHA-1

Title : Signing based on SHA-1

Problem Statement : Write a Python/ Java program to validate the parameter tuple for the security of the DSA. Design necessary classes. Use Miller-Rabin primality testing may be used.

Pre-requisites :

- (a) SHA-1 Algorithm

10.1 SHA-1

In cryptography, SHA-1 (Secure Hash Algorithm 1) is a cryptographic hash function designed by the United States National Security Agency and is a U.S. Federal Information Processing Standard published by the United States NIST. SHA-1 produces a 160-bit (20-byte) hash value known as a message digest. A SHA-1 hash value is typically rendered as a hexadecimal number, 40 digits long.

SHA-1 is no longer considered secure against well-funded opponents. In 2005, cryptanalysts found attacks on SHA-1 suggesting that the algorithm might not be secure enough for ongoing use,[3] and since 2010 many organizations have recommended its replacement by SHA-2 or SHA-3. Microsoft, Google and Mozilla have all announced that their respective browsers will stop accepting SHA-1 SSL certificates by 2017.

SHA-1 produces a message digest based on principles similar to those used by Ronald L. Rivest of MIT in the design of the MD4 and MD5 message digest algorithms, but has a more conservative design.

The original specification of the algorithm was published in 1993 under the title Secure Hash Standard, FIPS PUB 180, by U.S. government standards agency NIST (National Institute of Standards and Technology). This version is now often named SHA-0. It was withdrawn by the NSA shortly after publication and was superseded by the revised version, published in 1995 in FIPS PUB 180-1 and commonly designated SHA-1. SHA-1 differs from SHA-0 only by a single bitwise rotation in the message schedule of its compression function. According to the NSA, this was done to correct a flaw in the original algorithm which reduced its cryptographic security, but they did not provide any further explanation.[citation needed] Publicly available techniques did indeed compromise SHA-0 before SHA-1.

SHA-1 is one of the most secure hash algorithms. It is used in SSL (Secure Sockets Level), PGP (Pretty Good Privacy), XML Signatures, and in Microsoft's Xbox; the git source-code management system uses sha-1 hashes extensively, and it is used in hundreds of other applications (including from IBM, Cisco, Nokia, etc).

For a hash function for which L is the number of bits in the message digest, finding a message that corresponds to a given message digest can always be done using a brute force search in approximately 2^L evaluations. This is called a preimage attack and may or may not be practical depending on L and the particular computing environment. The second criterion, finding two different messages that produce the same message digest, namely a collision, requires on average only about $1.2 \cdot 2^{L/2}$ evaluations using a birthday attack. For the latter reason the strength of a hash function is usually compared to a symmetric cipher of half the message digest length. Thus SHA-1 was originally thought

to have 80-bit strength.

Cryptographers have produced collision pairs for SHA-0 and have found algorithms that should produce SHA-1 collisions in far fewer than the originally expected 280 evaluations.

In terms of practical security, a major concern about these new attacks is that they might pave the way to more efficient ones. Whether this is the case is yet to be seen, but a migration to stronger hashes is believed to be prudent. Some of the applications that use cryptographic hashes, like password storage, are only minimally affected by a collision attack. Constructing a password that works for a given account requires a preimage attack, as well as access to the hash of the original password, which may or may not be trivial. Reversing password encryption (e.g. to obtain a password to try against a user's account elsewhere) is not made possible by the attacks. (However, even a secure password hash can't prevent brute-force attacks on weak passwords.)

In the case of document signing, an attacker could not simply fake a signature from an existing document—the attacker would have to produce a pair of documents, one innocuous and one damaging, and get the private key holder to sign the innocuous document. There are practical circumstances in which this is possible; until the end of 2008, it was possible to create forged SSL certificates using an MD5 collision.

10.2 Mathematical Model

10.2.1 INPUT:

$I = C, S$

$C = M, P$

M is a message input where $|M| < 2^{64}-1$, processed in 512 bit blocks.

10.2.2 PROCESSING:

$F_n = \text{Request, Accept, Reply}$

F_{req} is the request to the server.

F_{acc} is when the server accepts the request

F_{rep} is the acknowledgment from the server

$M_p = P, L$

P is the padding operation where blocks are converted to 512 bit blocks by adding one 1 and other 0's

Padding = 1000...0L | where $L \in [0, 164]$ indicating M in binary

1 block = 512 bits = 16 words = $W_0P, \dots, W_{15}P$

$F = 0, 1160+5120, 1160$ through a series of 80 rounds of $v, +, -$ and rotate on words W_i and H_i

10.2.3 OUTPUT:

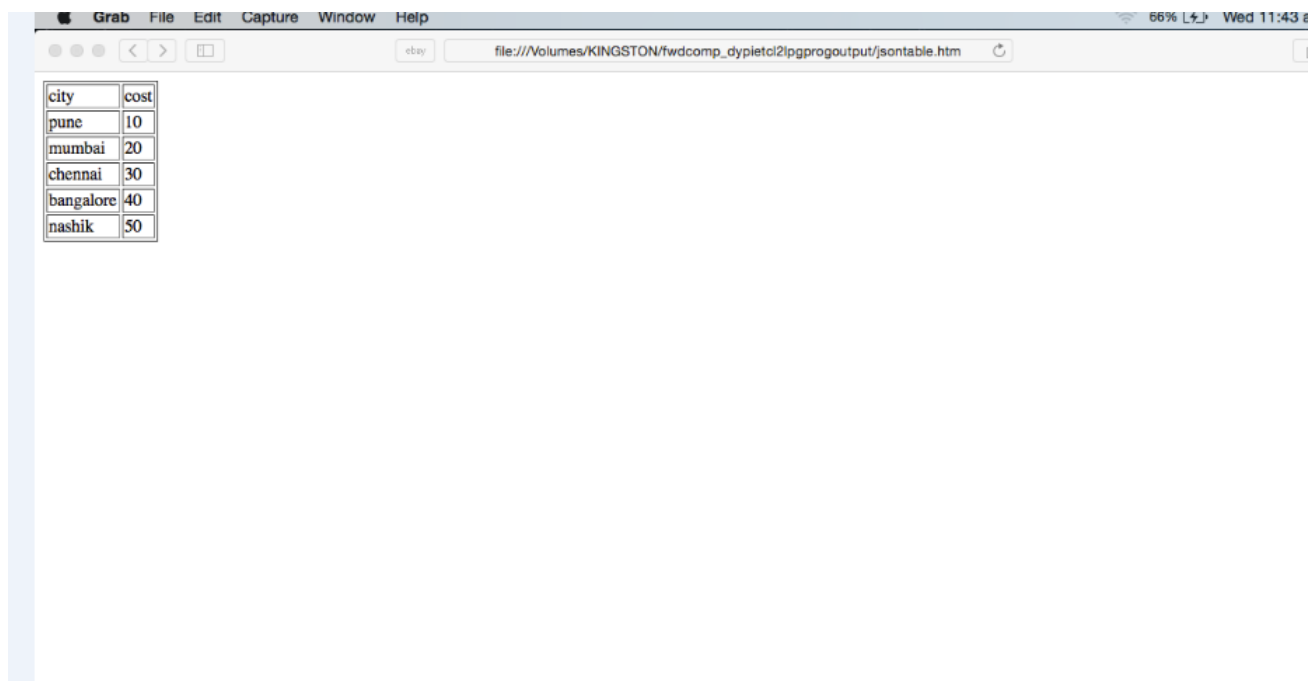
IV = a constant of 160 bits = 5 words = H0P.....H4P.

Resulting in a 160 bit digest. Which is fixed length output.

10.3 Testing

Sr.no	Tc Description	Tc i/p	Expected output	Actual output	Result
P#03	Check whether the inputs(Plaintext) are accepted	String plaintext	Valid input	Valid input	Pass
P#01	Check whether the Hash value matches	Hash value From hash function	True(authenticated)	True (authenticated)	Pass
N#01	Check whether the invalid Hash value matches	Hash value From Hash function	False (unauthenticated)	False (unauthenticated)	Pass
WB#01	Check for final Round(i) where $i < 65$	Round(i)	'i' for round(i) less than 65	'i' for round(i) less than 65	Pass
WB#02	Check whether for Correct Hash value message is sent	Hash value match	Message sent	Message sent	Pass
WB#03	Check whether for incorrect Hash value message is not sent	Hash value no match	Message sending terminated	Message sending terminated	Pass

FIGURE 10.1: Flow graph :Test cases of SHA1



The screenshot shows a web browser window with a table of city costs. The table has two columns: 'city' and 'cost'. The data rows are: pune (10), mumbai (20), chennai (30), bangalore (40), and nashik (50). The browser's address bar shows the file path: file:///Volumes/KINGSTON/fwdcomp_dyptel2/pgprogroutput/jsontable.htm.

city	cost
pune	10
mumbai	20
chennai	30
bangalore	40
nashik	50

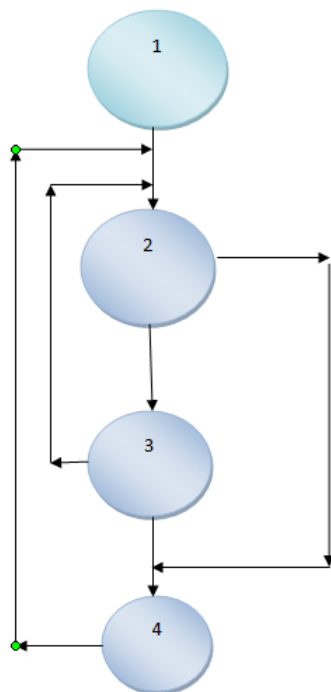
FIGURE 10.2

10.4 Testing

Test case id	Description	Input	Expected Result	Actual Result	Pass/Fail
BV#01	To check whether the list of cities and cost is displayed	List of cities with distances	Cities with the their cost displayed in ascending order	Cities with the their cost displayed in ascending order	Pass
P#01	To check whether the all cities have same distance	List of cities with same input	Erroneous input	Erroneous input	Fail
N#01	To check if city names and distances used are entered properly for costing	List of different cities and distances not mentioned in the code	Erroneous input	Erroneous input	Fail
WB#01	TO check if the cost of one city is less than other	Cost of 2 cities	Return the city having less cost	Return the city having less cost	Pass
WB#01	TO check if the cost of one city is greater than other	Cost of 2 cities	Return the city having greater cost	Return the city having greater cost	Pass

10.5 Conclusion

Thus a message is transmitted using network resources from one machine to another calculate and demonstrate the use of a Hash value equivalent to SHA-1.



Assignment 11

DSA signature

Title : DSA signature

Problem Statement : Write a program to produce a DSA signature using parameter tuple (p, q, g) , long term key pair and a message digest.

Pre-requisites :

(a) DSA Algorithm

Technologies Used :

11.1 Description:

A digital signature is a mathematical scheme for demonstrating the authenticity of a digital message or documents. A valid digital signature gives a recipient reason to believe that the message was created by a known sender, that the sender cannot deny having sent the message (authentication and non-repudiation), and that the message was not altered in transit (integrity). Digital signatures are a standard element of most cryptographic protocol suites, and are commonly used for software distribution, financial transactions, contract management software, and in other cases where it is important to detect forgery or tampering.

11.2 Explanation

Digital signatures are often used to implement electronic signatures, a broader term that refers to any electronic data that carries the intent of a signature,[1] but not all electronic signatures use digital signatures.[2][3] In some countries, including the United States, India, Brazil, Saudi Arabia,[4] the European Union and Switzerland,[5][6] electronic signatures have legal significance. Digital signatures employ asymmetric cryptography. In many instances they provide a layer of validation and security to messages sent through a nonsecure channel: Properly implemented, a digital signature gives the receiver reason to believe the message was sent by the claimed sender. Digital seals and signatures are equivalent to handwritten signatures and stamped seals.[7] Digital signatures are equivalent to traditional handwritten signatures in many respects, but properly implemented digital signatures are more difficult to forge than the handwritten type. Digital signature schemes, in the sense used here, are cryptographically based, and must be implemented properly to be effective. Digital signatures can also provide non-repudiation, meaning that the signer cannot successfully claim they did not sign a message, while also claiming their private key remains secret; further, some non-repudiation schemes offer a time stamp for the digital signature, so that even if the private key is exposed, the signature is valid. Digitally signed messages may be anything representable as a bitstring: examples include electronic mail, contracts, or a message sent via some other cryptographic protocol.

11.3 Description:

A digital signature scheme typically consists of three algorithms;

- (a) A key generation algorithm that selects a private key uniformly at random from a set of possible private keys. The algorithm outputs the private key and a corresponding public key.
- (b) A signing algorithm that, given a message and a private key, produces a signature.

- (c) A signature verifying algorithm that, given the message, public key and signature, either accepts or rejects the message's claim to authenticity.

Two main properties are required. First, the authenticity of a signature generated from a fixed message and fixed private key can be verified by using the corresponding public key. Secondly, it should be computationally infeasible to generate a valid signature for a party without knowing that party's private key. A digital signature is an authentication mechanism that enables the creator of the message to attach a code that acts as a signature. Formally, a digital signature scheme is a triple of probabilistic polynomial time algorithms, (G, S, V) , satisfying:

- (d) G (key-generator) generates a public key, pk , and a corresponding private key, sk , on input $1n$, where n is the security parameter.
- (e) S (signing) returns a tag, t , on the inputs: the private key, sk , and a string, x .
- (f) V (verifying) outputs accepted or rejected on the inputs: the public key, pk , a string, x , and a tag, t .

For correctness, S and V must satisfy $\Pr[(pk, sk) \leftarrow G(1n), V(pk, x, S(sk, x)) = \text{accepted}] = 1$. [8] A digital signature scheme is secure if for every non-uniform probabilistic polynomial time adversary, $\Pr[(pk, sk) \leftarrow G(1n), (x, t) \leftarrow AS(sk, \bullet)(pk, 1n), x \in Q, V(pk, x, t) = \text{accepted}] < \text{negl}(n)$, where $AS(sk, \bullet)$ denotes that A has access to the oracle, $S(sk, \bullet)$, and Q denotes the set of the queries on S made by A , which knows the public key, pk , and the security parameter, n . Note that we require any adversary cannot directly query the string, x , on S . [9]

11.4 How they work

To create RSA signature keys, generate an RSA key pair containing a modulus, N , that is the product of two large primes, along with integers, e and d , such that $d \equiv 1 \pmod{\phi(N)}$, where ϕ is the Euler phi-function. The signer's public key consists of N and e , and the signer's secret key contains d . To sign a message, m , the signer computes $m^d \pmod{N}$. To verify, the receiver checks that $e \cdot m \pmod{N}$. As noted earlier, this basic scheme is not very secure. To prevent attacks, one can first apply a cryptographic hash function to the message, m , and then apply the RSA algorithm described above to the result. This approach can be proven secure in the so-called random oracle model [clarification needed]. Most early signature schemes were of a similar type: they involve the use of a trapdoor permutation, such as the RSA function, or in the case of the Rabin signature scheme, computing square modulo composite, n . A trapdoor permutation family is a family of permutations, specified by a parameter, that is easy to compute in the forward direction, but is difficult to compute in the reverse direction without already knowing the private key ("trapdoor"). Trapdoor permutations can be used for digital signature schemes, where computing the reverse direction

with the secret key is required for signing, and computing the forward direction is used to verify signatures. Used directly, this type of signature scheme is vulnerable to a key-only existential forgery attack. To create a forgery, the attacker picks a random signature and uses the verification procedure to determine the message, m , corresponding to that signature.[19] In practice, however, this type of signature is not used directly, but rather, the message to be signed is first hashed to produce a short digest that is then signed. This forgery attack, then, only produces the hash function output that corresponds to $h(m)$, but not a message that leads to that value, which does not lead to an attack. In the random oracle model, this hash-then-sign form of signature is existentially unforgeable, even against a chosen-plaintext attack.[11][clarification needed] There are several reasons to sign such a hash (or message digest) instead of the whole document.

11.4.1 For efficiency

The signature will be much shorter and thus save time since hashing is generally much faster than signing in practice. For compatibility Messages are typically bit strings, but some signature schemes operate on other domains (such as, in the case of RSA, numbers modulo a composite number N). A hash function can be used to convert an arbitrary input into the proper format.

11.4.2 For integrity

Without the hash function, the text "to be signed" may have to be split (separated) in blocks small enough for the signature scheme to act on them directly. However, the receiver of the signed blocks is not able to recognize if all the blocks are present and in the appropriate order.

11.4.3 Notions of security

In their foundational paper, Goldwasser, Micali, and Rivest lay out a hierarchy of attack models against digital signatures:[18]

- In a key-only attack, the attacker is only given the public verification key.
- In a known message attack, the attacker is given valid signatures for a variety of messages known by the attacker but not chosen by the attacker.
- In an adaptive chosen message attack, the attacker first learns signatures on arbitrary messages of the attacker's choice. They also describe a hierarchy of attack results:[18]
- A total break results in the recovery of the signing key.
- A universal forgery attack results in the ability to forge signatures for any message.

- A selective forgery attack results in a signature on a message of the adversary's choice.
- An existential forgery merely results in some valid message/signature pair not already known to the adversary. The strongest notion of security, therefore, is security against existential forgery under an adaptive chosen message attack.

11.5 Applications of digital signatures

As organizations move away from paper documents with ink signatures or authenticity stamps, digital signatures can provide added assurances of the evidence to provenance, identity, and status of an electronic document as well as acknowledging informed consent and approval by a signatory. The United States Government Printing Office (GPO) publishes electronic versions of the budget, public and private laws, and congressional bills with digital signatures. Universities including Penn State, University of Chicago, and Stanford are publishing electronic student transcripts with digital signatures. Below are some common reasons for applying a digital signature to communications:

11.5.1 Authentication

Although messages may often include information about the entity sending a message, that information may not be accurate. Digital signatures can be used to authenticate the source of messages. When ownership of a digital signature secret key is bound to a specific user, a valid signature shows that the message was sent by that user. The importance of high confidence in sender authenticity is especially obvious in a financial context. For example, suppose a bank's branch office sends instructions to the central office requesting a change in the balance of an account. If the central office is not convinced that such a message is truly sent from an authorized source, acting on such a request could be a grave mistake.

11.5.2 Integrity

In many scenarios, the sender and receiver of a message may have a need for confidence that the message has not been altered during transmission. Although encryption hides the contents of a message, it may be possible to change an encrypted message without understanding it. (Some encryption algorithms, known as nonmalleable ones, prevent this, but others do not.) However, if a message is digitally signed, any change in the message after signature invalidates the signature. Furthermore, there is no efficient way to modify a message and its signature to produce a new message with a valid signature, because this is still considered to be computationally infeasible by most cryptographic hash functions (see collision resistance).

11.5.3 Non-repudiation

Non-repudiation, or more specifically non-repudiation of origin, is an important aspect of digital signatures. By this property, an entity that has signed some information cannot at a later time deny having signed it. Similarly, access to the public key only does not enable a fraudulent party to fake a valid signature. Note that these authentication, non-repudiation etc. properties rely on the secret key not having been revoked prior to its usage. Public revocation of a key-pair is a required ability, else leaked secret keys would continue to implicate the claimed owner of the key-pair. Checking revocation status requires an "online" check; e.g., checking a "Certificate Revocation List" or via the "Online Certificate Status Protocol". Very roughly this is analogous to a vendor who receives credit-cards first checking online with the credit-card issuer to find if a given card has been reported lost or stolen. Of course, with stolen key pairs, the theft is often discovered only after the secret key's use, e.g., to sign a bogus certificate for espionage purpose.

11.6 Mathematical Model

11.6.1 INPUT:

$I = C, S$

$C = m \mid$ where m is a message that is to be sent to the destination

$S = S_v \mid$ where S_v is a function to verify the signature

11.6.2 PROCESSING:

KEY GENERATION:

h = hash value of the message

$Pk = Pr, Pu \mid$ Pr is the private key and Pu is the public key of the client Where p and q are two prime nos. used to generate both keys $(p-1) \mid q \mid$ is the prime modulus

$g = h^{((p-1)/q)} \mid p.q \mid$, where $1 < g < p$, where $(g^{**q}) \mid p \mid = 1$ is g 's multiplicative order modulo $p \mid g \mid$

$x \mid 0 < x < q$ where $x \mid$

$y = (g^{**x}) \mid p \mid$

$Pu = (p, q, g, y)$

$Pr = (p, q, g, x)$

SIGNATURE GENERATION:

$S' = \text{Generation}$

Generation = $(h, k \mid$ where h is hash value and $0 < k < q$)

$r = ((g^{**k}) \mid p \mid) \mid q \mid$, if $r = 0$, then select different k

calculate I , such that $k \cdot i \mid q \mid = 1$

$s = i \cdot (h + r \cdot x)$, if $s = 0$, select a different k .

$Ds = r, s$

11.6.3 OUTPUT: $S = V$ $V = \text{Verification} = (((g^{*u_1})(y^{*u_2})) \bmod p) \bmod q,$

Where w is called the multiplicative inverse of s modulo q , $u_1 = h * w \bmod q$ and $u_2 = r * w \bmod q$

if $V = r$

Signature valid

11.7 Conclusion

Thus we have implemented a DSA signature using parameter.

Assignment 12

DSA signature

Title : DSA signature validation

Problem Statement : Write a Python/ Java program to validate the parameter tuple for the security of the DSA. Design necessary classes. Use Miller-Rabin primality testing may be used.

Pre-requisites :

(a) DSA Algorithm

Technologies Used :

12.1 Description:

A digital signature is a mathematical scheme for demonstrating the authenticity of a digital message or documents. A valid digital signature gives a recipient reason to believe that the message was created by a known sender, that the sender cannot deny having sent the message (authentication and non-repudiation), and that the message was not altered in transit (integrity). Digital signatures are a standard element of most cryptographic protocol suites, and are commonly used for software distribution, financial transactions, contract management software, and in other cases where it is important to detect forgery or tampering.

12.2 Explanation

Digital signatures are often used to implement electronic signatures, a broader term that refers to any electronic data that carries the intent of a signature,[1] but not all electronic signatures use digital signatures.[2][3] In some countries, including the United States, India, Brazil, Saudi Arabia,[4] the European Union and Switzerland,[5][6] electronic signatures have legal significance. Digital signatures employ asymmetric cryptography. In many instances they provide a layer of validation and security to messages sent through a nonsecure channel: Properly implemented, a digital signature gives the receiver reason to believe the message was sent by the claimed sender. Digital seals and signatures are equivalent to handwritten signatures and stamped seals.[7] Digital signatures are equivalent to traditional handwritten signatures in many respects, but properly implemented digital signatures are more difficult to forge than the handwritten type. Digital signature schemes, in the sense used here, are cryptographically based, and must be implemented properly to be effective. Digital signatures can also provide non-repudiation, meaning that the signer cannot successfully claim they did not sign a message, while also claiming their private key remains secret; further, some non-repudiation schemes offer a time stamp for the digital signature, so that even if the private key is exposed, the signature is valid. Digitally signed messages may be anything representable as a bitstring: examples include electronic mail, contracts, or a message sent via some other cryptographic protocol.

12.3 Description:

A digital signature scheme typically consists of three algorithms;

- (a) A key generation algorithm that selects a private key uniformly at random from a set of possible private keys. The algorithm outputs the private key and a corresponding public key.
- (b) A signing algorithm that, given a message and a private key, produces a signature.

- (c) A signature verifying algorithm that, given the message, public key and signature, either accepts or rejects the message's claim to authenticity.

Two main properties are required. First, the authenticity of a signature generated from a fixed message and fixed private key can be verified by using the corresponding public key. Secondly, it should be computationally infeasible to generate a valid signature for a party without knowing that party's private key. A digital signature is an authentication mechanism that enables the creator of the message to attach a code that acts as a signature. Formally, a digital signature scheme is a triple of probabilistic polynomial time algorithms, (G, S, V) , satisfying:

- (d) G (key-generator) generates a public key, pk , and a corresponding private key, sk , on input $1n$, where n is the security parameter.
- (e) S (signing) returns a tag, t , on the inputs: the private key, sk , and a string, x .
- (f) V (verifying) outputs accepted or rejected on the inputs: the public key, pk , a string, x , and a tag, t .

For correctness, S and V must satisfy $\Pr[(pk, sk) \leftarrow G(1n), V(pk, x, S(sk, x)) = \text{accepted}] = 1$. [8] A digital signature scheme is secure if for every non-uniform probabilistic polynomial time adversary, $\Pr[(pk, sk) \leftarrow G(1n), (x, t) \leftarrow AS(sk, \bullet)(pk, 1n), x \in Q, V(pk, x, t) = \text{accepted}] < \text{negl}(n)$, where $AS(sk, \bullet)$ denotes that A has access to the oracle, $S(sk, \bullet)$, and Q denotes the set of the queries on S made by A , which knows the public key, pk , and the security parameter, n . Note that we require any adversary cannot directly query the string, x , on S . [9]

12.4 How they work

To create RSA signature keys, generate an RSA key pair containing a modulus, N , that is the product of two large primes, along with integers, e and d , such that $d \equiv 1 \pmod{\phi(N)}$, where ϕ is the Euler phi-function. The signer's public key consists of N and e , and the signer's secret key contains d . To sign a message, m , the signer computes $m^d \pmod{N}$. To verify, the receiver checks that $e \cdot m \pmod{N}$. As noted earlier, this basic scheme is not very secure. To prevent attacks, one can first apply a cryptographic hash function to the message, m , and then apply the RSA algorithm described above to the result. This approach can be proven secure in the so-called random oracle model [clarification needed]. Most early signature schemes were of a similar type: they involve the use of a trapdoor permutation, such as the RSA function, or in the case of the Rabin signature scheme, computing square modulo composite, n . A trapdoor permutation family is a family of permutations, specified by a parameter, that is easy to compute in the forward direction, but is difficult to compute in the reverse direction without already knowing the private key ("trapdoor"). Trapdoor permutations can be used for digital signature schemes, where computing the reverse direction

with the secret key is required for signing, and computing the forward direction is used to verify signatures. Used directly, this type of signature scheme is vulnerable to a key-only existential forgery attack. To create a forgery, the attacker picks a random signature and uses the verification procedure to determine the message, m , corresponding to that signature.[19] In practice, however, this type of signature is not used directly, but rather, the message to be signed is first hashed to produce a short digest that is then signed. This forgery attack, then, only produces the hash function output that corresponds to $h(m)$, but not a message that leads to that value, which does not lead to an attack. In the random oracle model, this hash-then-sign form of signature is existentially unforgeable, even against a chosen-plaintext attack.[11][clarification needed] There are several reasons to sign such a hash (or message digest) instead of the whole document.

12.4.1 For efficiency

The signature will be much shorter and thus save time since hashing is generally much faster than signing in practice. For compatibility Messages are typically bit strings, but some signature schemes operate on other domains (such as, in the case of RSA, numbers modulo a composite number N). A hash function can be used to convert an arbitrary input into the proper format.

12.4.2 For integrity

Without the hash function, the text "to be signed" may have to be split (separated) in blocks small enough for the signature scheme to act on them directly. However, the receiver of the signed blocks is not able to recognize if all the blocks are present and in the appropriate order.

12.4.3 Notions of security

In their foundational paper, Goldwasser, Micali, and Rivest lay out a hierarchy of attack models against digital signatures:[18]

- In a key-only attack, the attacker is only given the public verification key.
- In a known message attack, the attacker is given valid signatures for a variety of messages known by the attacker but not chosen by the attacker.
- In an adaptive chosen message attack, the attacker first learns signatures on arbitrary messages of the attacker's choice. They also describe a hierarchy of attack results:[18]
- A total break results in the recovery of the signing key.
- A universal forgery attack results in the ability to forge signatures for any message.

- A selective forgery attack results in a signature on a message of the adversary's choice.
- An existential forgery merely results in some valid message/signature pair not already known to the adversary. The strongest notion of security, therefore, is security against existential forgery under an adaptive chosen message attack.

12.5 Applications of digital signatures

As organizations move away from paper documents with ink signatures or authenticity stamps, digital signatures can provide added assurances of the evidence to provenance, identity, and status of an electronic document as well as acknowledging informed consent and approval by a signatory. The United States Government Printing Office (GPO) publishes electronic versions of the budget, public and private laws, and congressional bills with digital signatures. Universities including Penn State, University of Chicago, and Stanford are publishing electronic student transcripts with digital signatures. Below are some common reasons for applying a digital signature to communications:

12.5.1 Authentication

Although messages may often include information about the entity sending a message, that information may not be accurate. Digital signatures can be used to authenticate the source of messages. When ownership of a digital signature secret key is bound to a specific user, a valid signature shows that the message was sent by that user. The importance of high confidence in sender authenticity is especially obvious in a financial context. For example, suppose a bank's branch office sends instructions to the central office requesting a change in the balance of an account. If the central office is not convinced that such a message is truly sent from an authorized source, acting on such a request could be a grave mistake.

12.5.2 Integrity

In many scenarios, the sender and receiver of a message may have a need for confidence that the message has not been altered during transmission. Although encryption hides the contents of a message, it may be possible to change an encrypted message without understanding it. (Some encryption algorithms, known as nonmalleable ones, prevent this, but others do not.) However, if a message is digitally signed, any change in the message after signature invalidates the signature. Furthermore, there is no efficient way to modify a message and its signature to produce a new message with a valid signature, because this is still considered to be computationally infeasible by most cryptographic hash functions (see collision resistance).

12.5.3 Non-repudiation

Non-repudiation, or more specifically non-repudiation of origin, is an important aspect of digital signatures. By this property, an entity that has signed some information cannot at a later time deny having signed it. Similarly, access to the public key only does not enable a fraudulent party to fake a valid signature. Note that these authentication, non-repudiation etc. properties rely on the secret key not having been revoked prior to its usage. Public revocation of a key-pair is a required ability, else leaked secret keys would continue to implicate the claimed owner of the key-pair. Checking revocation status requires an "online" check; e.g., checking a "Certificate Revocation List" or via the "Online Certificate Status Protocol". Very roughly this is analogous to a vendor who receives credit-cards first checking online with the credit-card issuer to find if a given card has been reported lost or stolen. Of course, with stolen key pairs, the theft is often discovered only after the secret key's use, e.g., to sign a bogus certificate for espionage purpose.

12.6 Miller-Rabin Test

The Miller–Rabin primality test or Rabin–Miller primality test is a primality test: an algorithm which determines whether a given number is prime, similar to the Fermat primality test and the Solovay–Strassen primality test. Its original version, due to Gary L. Miller, is deterministic, but the determinism relies on the unproven Extended Riemann hypothesis; Michael O. Rabin modified it to obtain an unconditional probabilistic algorithm.

The algorithm can be written in pseudocode as follows:

Input 1: $n > 3$, an odd integer to be tested for primality;

Input 2: k , a parameter that determines the accuracy of the test

Output: composite if n is composite, otherwise probably prime

write $n - 1$ as $2^r \cdot d$ with d odd by factoring powers of 2 from $n - 1$

WitnessLoop: repeat k times:

pick a random integer a in the range $[2, n - 2]$

$x \leftarrow a^d \bmod n$

if $x = 1$ or $x = n - 1$ then

continue WitnessLoop

repeat $r - 1$ times:

$x \leftarrow x^2 \bmod n$

if $x = 1$ then

return composite

if $x = n - 1$ then

continue WitnessLoop

return composite

return probably prime

Using modular exponentiation by repeated squaring, the running

time of this algorithm is $O(k \log^3 n)$, where k is the number of different values of a that we test; thus this is an efficient, polynomial-time algorithm. FFT-based multiplication can push the running time down to $O(k \log^2 n \log \log n \log \log \log n) = \tilde{O}(k \log^2 n)$.

12.7 Mathematical Model

12.7.1 INPUT:

Let input set,

$I = \{q, i, n \mid q \text{ belongs to } N, i \text{ belongs to } N, n \text{ belongs to } n\}$

12.7.2 PROCESSING:

Functions:

F1: Miller Rabin primality testing

$$x = \sum_{x=0}^n x^2 \bmod n$$

12.7.3 OUTPUT:

Let output set O = result if prime no. or composite no.

12.8 Testing

Test case Id	Test case description	Input	Expected Output	Actual Output	Result
P#01	Positive large prime number is given as 1 st tuple.	Positive Integer value	Result of Miller Rabin primality testing	Result of Miller Rabin primality testing	Pass
N#01	Negative large prime number is given as 1 st tuple.	Negative Integer value	Error Message	Error Message	Pass
P#02	Check the iteration number for testing	Positive iteration number	Perform Miller Rabin Testing	Perform Miller Rabin Testing	Pass
N#02	Check the iteration number for testing	Negative Integer value	Error Message	Error Message	Pass

FIGURE 12.1: Flow graph :Test cases of DSA

12.9 Conclusion

Thus we have validated DSA signature parameters using Miller-Rabin primality testing.

Assignment 13

Intrusion Detection System

Title : Installation of Intrusion Detection System

Problem Statement : Install and Use Latest IDS (Open Source).

Pre-requisites :

(a) Intrusion Detection System

13.1 Intrusion Detection System

An intrusion detection system (IDS) is a device or software application that monitors network or system activities for malicious activities or policy violations and produces electronic reports to a management station. IDS come in a variety of "flavors" and approach the goal of detecting suspicious traffic in different ways. There are network based (NIDS) and host based (HIDS) intrusion detection systems. NIDS is a network security system focusing on the attacks that come from the inside of the network (authorized users). Some systems may attempt to stop an intrusion attempt but this is neither required nor expected of a monitoring system. Intrusion detection and prevention systems (IDPS) are primarily focused on identifying possible incidents, logging information about them, and reporting attempts. In addition, organizations use IDPSes for other purposes, such as identifying problems with security policies, documenting existing threats and deterring individuals from violating security policies. IDPSes have become a necessary addition to the security infrastructure of nearly every organization.

IDPSes typically record information related to observed events, notify security administrators of important observed events and produce reports. Many IDPSes can also respond to a detected threat by attempting to prevent it from succeeding. They use several response techniques, which involve the IDPS stopping the attack itself, changing the security environment (e.g. reconfiguring a firewall) or changing the attack's content.

13.2 Types of IDS

- (a) Network Intrusion Detection Systems (NIDS)
- (b) Host based Intrusion Detection Systems (HIDS)

13.2.1 Network Intrusion Detection Systems (NIDS)

Network Intrusion Detection Systems (NIDS) are placed at a strategic point or points within the network to monitor traffic to and from all devices on the network. It performs an analysis of passing traffic on the entire subnet, and matches the traffic that is passed on the subnets to the library of known attacks. Once an attack is identified, or abnormal behavior is sensed, the alert can be sent to the administrator. An example of an NIDS would be installing it on the subnet where firewalls are located in order to see if someone is trying to break into the firewall. Ideally one would scan all inbound and outbound traffic, however doing so might create a bottleneck that would impair the overall speed of the network. OPNET and NetSim are commonly used tools for simulation network intrusion detection systems. NID Systems are also capable of comparing signatures for similar packets to link and

drop harmful detected packets which have a signature matching the records in the NIDS. When we classify the designing of the NIDS according to the system interactivity property, there are two types: on-line and off-line NIDS. On-line NIDS deals with the network in real time. It analyses the Ethernet packets and applies some rules, to decide if it is an attack or not. Off-line NIDS deals with stored data and passes it through some processes to decide if it is an attack or not

13.2.2 Host based Intrusion Detection Systems (HIDS)

Host Intrusion Detection Systems (HIDS) run on individual hosts or devices on the network. A HIDS monitors the inbound and outbound packets from the device only and will alert the user or administrator if suspicious activity is detected. It takes a snapshot of existing system files and matches it to the previous snapshot. If the critical system files were modified or deleted, an alert is sent to the administrator to investigate. An example of HIDS usage can be seen on mission critical machines, which are not expected to change their configurations.

13.3 Passive and reactive systems

In a passive system, the intrusion detection system (IDS) sensor detects a potential security breach, logs the information and signals an alert on the console or owner. In a reactive system, also known as an intrusion prevention system (IPS), the IPS auto-responds to the suspicious activity by resetting the connection or by reprogramming the firewall to block network traffic from the suspected malicious source. The term IDPS is commonly used where this can happen automatically or at the command of an operator; systems that both "detect (alert)" and "prevent".

13.4 Comparison with firewalls

Though they both relate to network security, an intrusion detection system (IDS) differs from a firewall in that a firewall looks outwardly for intrusions in order to stop them from happening. Firewalls limit access between networks to prevent intrusion and do not signal an attack from inside the network. An IDS evaluates a suspected intrusion once it has taken place and signals an alarm. An IDS also watches for attacks that originate from within a system. This is traditionally achieved by examining network communications, identifying heuristics and patterns (often known as signatures) of common computer attacks, and taking action to alert operators. A system that terminates connections is called an intrusion prevention system, and is another form of an application layer firewall.

13.5 Statistical anomaly and signature-based ID-Ses

All Intrusion Detection Systems use one of two detection techniques:

13.5.1 Statistical anomaly-based IDS

An IDS which is anomaly based will monitor network traffic and compare it against an established baseline. The baseline will identify what is "normal" for that network- what sort of bandwidth is generally used, what protocols are used that it may raise a False Positive alarm for a legitimate use of bandwidth if the baselines are not intelligently configured.

13.5.2 Signature-based IDS

A signature based IDS will monitor packets on the network and compare them against a database of signatures or attributes from known malicious threats. This is similar to the way most antivirus software detects malware. The issue is that there will be a lag between a new threat being discovered in the wild and the signature for detecting that threat being applied to the IDS. During that lag time the IDS would be unable to detect the new threat.

13.6 Limitations

- (a) Noise can severely limit an intrusion detection system's effectiveness. Bad packets generated from software bugs, corrupt DNS data, and local packets that escaped can create a significantly high false-alarm rate.
- (b) It is not uncommon for the number of real attacks to be far below the number of false-alarms. Number of real attacks is often so far below the number of false-alarms that the real attacks are often missed and ignored.
- (c) Many attacks are geared for specific versions of software that are usually outdated. A constantly changing library of signatures is needed to mitigate threats. Outdated signature databases can leave the IDS vulnerable to newer strategies.
- (d) For signature-based IDSes there will be lag between a new threat discovery and its signature being applied to the IDS. During this lag time the IDS will be unable to identify the threat.
- (e) It cannot compensate for a weak identification and authentication mechanisms or for weaknesses in network protocols. When an attacker gains access due to weak authentication mechanism then IDS cannot prevent the adversary from any malpractise.

- (f) Encrypted packets are not processed by the intrusion detection software. Therefore, the encrypted packet can allow an intrusion to the network that is undiscovered until more significant network intrusions have occurred.
- (g) Intrusion detection software provides information based on the network address that is associated with the IP packet that is sent into the network. This is beneficial if the network address contained in the IP packet is accurate. However, the address that is contained in the IP packet could be faked or scrambled.
- (h) Due to the nature of NIDS systems, and the need for them to analyse protocols as they are captured, NIDS systems can be susceptible to same protocol based attacks that network hosts may be vulnerable. Invalid data and TCP/IP stack attacks may cause an NIDS to crash.

13.7 Snort

Snort is a free and open source network intrusion prevention system (NIPS) and network intrusion detection system (NIDS) created by Martin Roesch in 1998. Snort is now developed by Sourcefire, of which Roesch is the founder and CTO, and which has been owned by Cisco since 2013.

In 2009, Snort entered InfoWorld's Open Source Hall of Fame as one of the "greatest [pieces of] open source software of all time".

13.7.1 Introduction

An intrusion detection system (IDS) inspects all inbound and outbound network activity and identifies suspicious patterns that may indicate a network or system attack from someone attempting to break into or compromise a system.

An IDS differs from a firewall in that a firewall inspects the traffic and stops it based upon user specified rules. An IDS on the other hand, inspects and evaluates the traffic to determine if it is suspicious. The IDS may raise alerts based upon the analysis.

There are multiple locations an IDS should be located. The following are two example placement locations.

Our first example, shows the IDS behind our firewall. Data coming into the Local Area Network (LAN) is mirrored to the port the IDS is connected to. The interface on the IDS is in promiscuous mode allowing it to inspect all traffic.

13.7.2 Uses

- (a) Snort's open source network-based intrusion detection system (NIDS) has the ability to perform real-time traffic analysis and packet logging on Internet Protocol (IP) networks. Snort performs protocol analysis, content searching and matching. These basic services have many purposes including application-aware

- triggered quality of service, to de-prioritize bulk traffic when latency-sensitive applications are in use.
- (b) The program can also be used to detect probes or attacks, including, but not limited to, operating system fingerprinting attempts, common gateway interface, buffer overflows, server message block probes, and stealth port scans.
 - (c) Snort can be configured in three main modes: sniffer, packet logger, and network intrusion detection. In sniffer mode, the program will read network packets and display them on the console. In packet logger mode, the program will log packets to the disk. In intrusion detection mode, the program will monitor network traffic and analyze it against a rule set defined by the user. The program will then perform a specific action based on what has been identified.

13.8 Installation of Snort

13.8.1 Pre-requisites

Snort has four main pre-requisites:

- (a) pcap
- (b) PCRE
- (c) Libdnet
- (d) DAQ

First we want to install all the tools required for building software. The build-essentials package does this for us

```
sudo apt-get install -y build-essential
```

Once our build tools are installed, we install all Snort pre-requisites

that are available from the Ubuntu repositories

```
sudo apt-get install -y libpcap-dev libpcre3-dev libdumbnet-dev
```

We will be downloading a number of tarballs for various software

packages. We will create a folder called snort src to keep them all in one place: `mkdir /snortsrc`

```
cd /snortsrc
```

The Snort DAQ (Data Acquisition library) has a few pre-requisites

that need to be installed

```
sudo apt-get install -y bison flex
```

Download and install the latest version of DAQ from the Snort

website. The steps below use `wget` to download version 2.0.6 of DAQ, which is the latest version at the time of writing this guide.

```
cd /snortsrc
```

```
wget https://www.snort.org/downloads/snort/daq-2.0.6.tar.gz
tar -xvzf daq-2.0.6.tar.gz
cd daq-2.0.6
./configure

make
sudo make install
```

13.8.2 Installing Snort

To install Snort on Ubuntu, there is one additional required prerequisite that needs to be installed that is not mentioned in the documentation: `zlibg` which is a compression library.

There are three optional libraries that improves functionality: `liblzma-dev` which provides decompression of swf files (adobe flash), `openssl`, and `libssl-dev` which both provide SHA and MD5 file signatures.

```
sudo apt-get install -y zlib1g-dev liblzma-dev openssl libssl-dev
```

We are now ready to download the Snort source tarball, compile, and then install. The `--enable-sourcefire` option gives Packet Performance Monitoring (PPM) 4 5 , which lets us do performance monitoring for rules and pre-processors, and builds Snort the same way that the Snort team does

```
cd /snortsrc
wget https://snort.org/downloads/snort/snort-2.9.8.0.tar.gz
tar -xvzf snort-2.9.8.0.tar.gz
cd snort-2.9.8.0
./configure --enable-sourcefire

make
```

```
sudo make install
```

Run the following command to update shared libraries (you'll get an error when you try to run Snort if you skip this step)

```
sudo ldconfig
```

Place a symlink to the Snort binary in `/usr/sbin`

```
sudo ln -s /usr/local/bin/snort /usr/sbin/snort
```

Test Snort by running the binary as a regular user, passing it the `-V` flag (which tells Snort to verify itself and any configuration files passed to it). You should see output similar to what is shown below (although exact version numbers may be slightly different):

```
user@snortserver: snort -V
-*> Snort! <*-
```

Version 2.9.8.0 GRE (Build 229)

By Martin Roesch The Snort Team: <http://www.snort.org/contactteam>

Copyright (C) 2014-2015 Cisco and/or its affiliates. All rights reserved.

Copyright (C) 1998-2013 Sourcefire, Inc., et al.

Using libpcap version 1.5.3

Using PCRE version: 8.31 2012-07-06

Using ZLIB version: 1.2.8

user@snortserver:

Configuring Snort to Run in NIDS Mode:

Since we don't want Snort to run as root, we need to create an unprivileged account and group for the daemon to run under (snort:snort). We will also create a number of files and directories required by Snort, and set permissions on those files. Snort will have the following directories: Configurations and rule files in /etc/snort Alerts will be written to /var/log/snort Compiled rules (.so rules) will be stored in /usr/local/lib/snort dynamicrules

Create the snort user and group:

```
sudo groupadd snort
sudo useradd snort -r -s /sbin/nologin -c SNORT_IDS -g snort
```

Create the Snort directories:

```
sudo mkdir /etc/snort
sudo mkdir /etc/snort/rules
sudo mkdir /etc/snort/rules/iplists
sudo mkdir /etc/snort/preprocrules
sudo mkdir /usr/local/lib/snortdynamicrules
sudo mkdir /etc/snort/sorules
```

Create some files that stores rules and ip lists

```
sudo touch /etc/snort/rules/iplists/blacklist.rules
sudo touch /etc/snort/rules/iplists/whitelist.rules
sudo touch /etc/snort/rules/local.rules
sudo touch /etc/snort/sid-msg.map
```

Create our logging directories

```
sudo mkdir /var/log/snort
sudo mkdir /var/log/snort/archivedlogs
```

Adjust permissions

```
sudo chmod -R 5775 /etc/snort
sudo chmod -R 5775 /var/log/snort
sudo chmod -R 5775 /var/log/snort/archivedlogs
```



```
sudo chmod -R 5775 /etc/snort/sorules  
sudo chmod -R 5775 /usr/local/lib/snortdynamicrules
```

Other open source IDS

- (a) Suricata
- (b) Bro
- (c) Kismet
- (d) OSSEC
- (e) Samhain
- (f) OpenDLP
- (g) FIM Only