**Experiment No.: Date:**

|  |  |
| --- | --- |
| **Aim:** | Squre and multiplt algorithm for exponential modulus calculation |
| **Apparatus:** | MATLAB 6.5/7, Computer set up |
| **Pre-Requisites:** | Concept of CRYPTOGAPHY, Modular arithmetic, |
| **Theory:** | Most computer languages do not have an operator that can efficiently compute exponentiation, particularly when the exponent is very large.  To make this type of calculation more efficient we need algorithms that are faster. Fast exponentiation is possible using square and multiplies method.  The main idea behind this method is to treat the exponent as a binary number.      Figure given below shows the process for calculating *y* = *ax* using the square and multiply algorithm. In this case, *exponents* = 22 = (10110)2 in binary. The exponent has five bits.  Demonstration of calculation of a22 using square-and-multiply method    Modular Exponentiation is used in cryptography. We consider an algorithm for calculating modular powers. The modular exponentiation problem is to compute  **a^x mod n**, given a, x, and n. The obvious algorithm multiplies a together x times. But there is a much faster algorithm which uses at most 2 log2 A multiplications. The algorithm uses the fact that one can reduce modulo n at each and every point.  i.e. ,  **(a\*b) mod n =( (a mod n) \* (b mod n) ) \* mod n**.    Example:  . |
| **Procedure:** | 1. 1. Write a function to calculate exponential mod (ax mod n) using square and multiply method. Call the function from the main program. 2. In the main program take a ,x andna as input from the user. 3. In function, convert the exponent x to binary and get its string length. 4. Create an array of power mods and store the number ‘a’ as its first element. Then square it, reduce it modulo n and store in the array as next entry. Repeat till string length is exhausted. 5. Display the array of powers modulo n. 6. Again read the binary string and power array. If the bit is 1, take its product modulo n of the corresponding location in power array otherwise skip. Repeat till string length is exhausted. Last product reduced modulo n is the required answer. Pass it to the main function. |
| **Result and Discussion:** |  |