|  |  |
| --- | --- |
| Cairo University  Faculty of Computers and artificial intelligence  Information Technology Department  IT423 - Information and Computers Networks Security  Fall 2023 |  |

**Lab Assignment (4)**

**Assignment deadline: - Monday** 3 January 2024 (All groups)

**Implement** **RSA Algorithm encryption and decryption**

**Input: -**

1-Plain text of any size 🡪 each character is changed to ASCII (***X***) and encrypted

**Output: -**

1- Cipher text in ASCII (decimal) for each character (***X***)

2- Selected p,q for each character (***X***)

3- Generated keys e,d for each character (***X***)

**Read the hints which are mentioned through this document carefully.**

**Grading Criteria: - (Total mark 15)**

1. **Encryption**🡪 **1 mark**
2. **Square and multiply algorithm** 🡪 **2 marks**
3. **Key generation** 🡪 **7.5 marks**
   1. Select two prime numbers p, q (p ≠ q) using fermat algorithm🡪 3 marks
   2. Compute n 🡪 0.5 mark
   3. Compute ø (n) 🡪 0.5 mark
   4. Select e 🡪 0.5 mark
   5. Compute d using extended euclid🡪 3 marks
4. **Display generated keys e & d 🡪0.5 mark**
5. **Display p, q 🡪0.5 mark**
6. **Decryption**🡪 **3.5 mark**
   1. Transforming into the CRT domain 🡪1 mark
   2. Computation in the CRT domain 🡪1 mark
   3. Inverse transformation of the result 🡪1.5 mark

**Remember:-**

***Hint1:-*To do any exponentiation (xH mod n) use Square and multiply algorithm as follows:-**

****

**Example: -** Compute x 26 using square and multiply algorithm

****

****

***Hint2: -***

* **For step 1 in key generation p & q must satisfy the following:-**

1. p, q are random numbers *X* < p, q <(where *X* is the ASCII (decimal) of one character in plaintext)
2. p, q are prime numbers (Test primality of them using Fermat Primality Test mentioned below )

* **For step 4 and 5 in key generation use Extended Euclidean Algorithm OR Binary Extended Euclidean Algorithm**



**mod p**

***Hint3: -* Step 1: s = 100**

**Step 1.2: is computed by using the square-and-multiply algorithm**

**RSA Encryption:**

****

***Hint4: -* Compute Exponentiation (xe mod n) using Square and Multiply method**

**RSA Decryption with the Chinese Remainder Theorem (CRT)**

We cannot choose a short private key without compromising the security for RSA as an attacked can use brute force attack.

In CRT the goal is to perform the exponentiation mod *n* efficiently. First we note that the party who possesses the private key also knows the primes *p* and *q*. The basic idea of the CRT is that rather than doing arithmetic with one “long” modulus *n*, we do two individual exponentiations modulo the two “short” primes *p* and *q*. This is a type of transformation arithmetic. Like any transform, there are three steps:

1. Transforming into the CRT domain
2. Computation in the CRT domain
3. Inverse transformation of the result

* **Transformation of the Input into the CRT Domain**

We simply reduce the base element *x* modulo the two factors *p* and *q* of the modulus *n*, and obtain what is called the modular representation of *x*.



* **Exponentiation in the CRT Domain**

With the reduced versions of x we perform the following two exponentiations:



where the two new exponents are given by:



* **Inverse Transformation into the Problem Domain**

The remaining step is now to assemble the final result *y* from its modular representation (*yp,yq*). This follows from the CRT and can be done as:



where the coefficients *cp* and *cq* are computed as:



**Example:-**

****

We now compute an RSA decryption for the ciphertext *y* = 15 using the CRT, i.e., the value= mod 143.

In the first step, we compute the modular representation of *y*:-



In the second step, we perform the exponentiation in the transform domain with the short exponents. These are:-



Here are the exponentiations:-



In the last step, we have to compute x from its modular representation (xp, xq). For this, we need the coefficients:-



***Hint5:-***

1. **Compute exponentiation (in example: and)using square and multiply method.**
2. **Compute cp and cq using EEA OR Binary EEA.**

The plaintext *x* follows now as:-

