Network Security
Session: July-Dec 2020
LAB Assignment
Deadline for Submission: 07th October, 2020, Wednesday, Midnight (Hard)

<u>NOTE:</u> All programs must be coded in **MIRACL**

1) Implement the **Euclidean algorithm** and find the greatest common divisor (GCD) of two randomly generated 512-bit numbers. Output the two numbers and their GCD to the terminal.

2) Implement **Extended Euclidean algorithm** and find multiplicative inverse of a randomly generated big number modulo a randomly generated 512-bit number. Test the existence of inverse before finding the inverse. Output the number, its inverse (if it exists) and the modulus to the terminal.

3) Implement **Square-and-Multiply algorithm** and compute modular exponentiation of a randomly generated big number, raised to the power of another randomly generated big number and modulo a randomly generated 512-bit number. Output the two numbers, the modulus and the results of modular exponentiation to the terminal.

4) Implement **Miller-Rabin primality testing algorithm** and use it to test the primeness of a randomly generated 512-bit number. Output the number, the total runs required to test the number, and its status - whether the number is composite or probably prime - to the terminal. Repeatedly run the test for different random numbers until a prime number is found.

5) Re-design the **key generation algorithm** of RSA (demonstrated during lab session) and generate prime numbers using Miller-Rabin algorithm implemented in **4),** and compute multiplicative inverses using Extended Euclidean algorithm implemented in **2)**.

6) The *textbook RSA encryption* (demonstrated during lab session) is insecure as the encryption is *deterministic* and *malleable*. To overcome these insecurities, **Optimal Asymmetric Encryption Padding** (OAEP) is used which introduces an element of randomness during encryption, therefore, converts textbook RSA to *probabilistic non-malleable* RSA. Modify encryption and decryption algorithms of RSA by incorporating OAEP encoding and decoding, respectively. Maintain following files during encryption:
   a) A2B.txt → Stores the user inputted ASCII plaintext in binary format
   b) B2O.txt → Stores the hexadecimal data obtained after OAEP encoding.
   c) O2C.txt → Stores the ciphertext in hexadecimal format.

   Maintain following files during decryption:
   a) C2O.txt → Stores the hexadecimal data after decryption of ciphertext.
   b) O2B.txt → Stores the binary data after OAEP decoding.

Finally, output the plaintext in O2B.txt to the terminal in ASCII format. Key size (i.e., the number of bits in N) should be at least 1024-bits.

7) Implement **Elgamal encryption**. Generate 1024-bit prime modulus using Miller-Rabin algorithm implemented in **4)**, use square-and-multiply algorithm implemented in **3)** to compute modular exponentiation and use Extended Euclidean algorithm implemented in **2)** to find multiplicative inverses.

8) Implement **Diffie-Hellman Key Exchange protocol**. Generate 1024-bit prime modulus using Miller-Rabin algorithm implemented in **4)**, use square-and-multiply algorithm implemented in **3)** to compute modular exponentiation and use Extended Euclidean algorithm implemented in **2)** to find multiplicative inverses.

*********END*********

Sources:

1. Euclidean Algorithm: https://en.wikipedia.org/wiki/Euclidean_algorithm
2. Extended Euclidean Algorithm:https://en.wikipedia.org/wiki/Extended_Euclidean_algorithm
3. Square-and-Multiply Algorithm: https://www.khanacademy.org/computing/computer-science/cryptography/modarithmetic/a/fast-modular-exponentiation
4. Miller-Rabin Primality Testing: https://en.wikipedia.org/wiki/Miller%E2%80%93Rabin_primality_test
5. Optimal Asymmetric Encryption Padding: https://en.wikipedia.org/wiki/Optimal_asymmetric_encryption_padding
6. Elgamal Encryption: http://homepages.math.uic.edu/~leon/mcs425-s08/handouts/el-gamal.pdf
7. Diffie-Hellman Key Exchange: https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange