Fall 2024

CS454/654 Reliability and Security of Computing Systems

Implementing RSA Encryption and Decryption

Homework 3

Due Date: Follow WebCampus Calendar **Total Marks**: 100

Objective: Complete the RSA encryption and decryption program using the given *skeleton_rsa.py* file. Your implementation should follow the modular design specified in the skeleton, ensuring each function operates as intended. Do not delete or modify any of the provided code structures; however, you may add new helper functions or additional code if needed.

Instructions

1. gcd Function (5 Marks)

Implement the **gcd(a, b)** function using the Euclidean algorithm.

Criteria: The function should return the greatest common divisor of a and b.

2. extended_gcd Function (10 Marks)

Implement the **extended_gcd(a, b)** function using the Euclidean algorithm.

Criteria: The function should return the gcd(a, b), x and y. For better understanding you can go through the algorithm and implementation from this link.

3. Miller-Rabin Primality Test (15 Marks)

Implement the **miller_rabin(n, k=40)** function for primality testing.

Criteria: This function should return True if n is likely a prime number; otherwise, return False. Use k iterations to improve the accuracy of the test. See chapter 2 section 6 for details.

4. Prime Generation (10 Marks)

Implement **generate_large_prime(bits)**, which generates a prime number with the specified bit length

Criteria: Use **random number generator** with the **seed** specified in the **main function** and then perform Miller-Rabin test within this function to verify primality.

5. Calculate Totient (5 Marks)

Implement **calculate_totient(p, q)**, which calculates the totient.

6. Choose Public Exponent e (5 Marks)

Implement **choose_e(totient)**, which selects an integer e such that **1<*e*<totient** and

gcd(e, totient) = 1.

Criteria: Ensure **e** meets the conditions and is suitable for encryption.

7. Calculate Private Exponent d (15 Marks)

Implement **calculate_d(e, totient)**, which calculates the modular inverse of e under totient.

Criteria: Use the Extended Euclidean Algorithm to find the correct value of d.

Remember: The extended_gcd function returns three values—**gcd(e, totient), x,** and **y.** If **gcd(e, totient) != 1**, you need to choose a different **e** and calculate the value of **d** again. However, if **gcd(e, totient) = 1**, then **(x % totient)** will give you the correct value of d.

8. Key Pair Generation (5 Marks)

Complete the **generate_keypair(bits, seed=None)** function to generate public and private key pairs.

Criteria: Ensure the function generates keys **(e, n)** and **(d, n)** and prints intermediate values for **p, q, n, totient, e,** and **d**.

9. Modular Exponentiation (10 Marks)

Implement **mod_exp(base, exp, mod)**, an efficient way to calculate

Criteria: This function should handle large exponents efficiently. For better understanding, go through this link.

10. RSA Encryption (10 Marks)

Complete **rsa_encrypt(message, e, n)** to encrypt a message.

Criteria: Convert each character to its ASCII value (You can use **ord(char)** in python), encrypt it, and store it in a list. Here, each character will be counted as a message block, M. This list of numbers is considered as the ciphertext of the original message.

11. RSA Decryption (10 Marks)

Complete **rsa_decrypt(ciphertext, d, n)** to decrypt a ciphertext back to the original message.

Criteria: Convert each encrypted character back to its ASCII representation, reconstructing the original message.

12. Main Function and Testing

The **main** function should be able to run your code and make sure your code can handle any bit lengths when generating two prime numbers, **p** and **q**. The main function has three arguments and these are

bits = number of bits of each generated prime number

seed = random number generating seed

message = original message

Sample Input

bits = 8

seed = 12

message = "Hello, RSA!"

Sample Output

Generated values for RSA key pair:

p (prime): 131, q (prime): 227, n (modulus): 29737, Totient (φ(n)): 29380

e (public exponent): 9063, d (private exponent): 17687

Public Key: (9063, 29737)

Private Key: (17687, 29737)

Original Message: Hello, RSA!

Ciphertext: [28007, 4906, 27815, 27815, 1800, 20894, 23604, 16239, 23167, 14334, 28701]

Decrypted Message: Hello, RSA!

RSA Encryption and Decryption successful!

Submission Instructions: Rename your script to *rsa.py* and upload your *.py file* to WebCampus for submission.

Note: Do not remove any code from the skeleton file. Follow the structure and add additional helper functions if necessary. Ensure that your code is efficient and correctly performs each operation as specified.