Cryptography and Network Security Lab

Assignment 8

Student Details

Name : Krunal Rank
Adm No. : U18CO081

1

```
# sample text : once upon a time there was a little girl named goldilocks she went
for a walk in the forest pretty soon she came upon a house she knocked and when no
one answered she walked right in at the table in the kitchen there were three bowls
of porridge goldilocks was hungry she tasted the porridge from the first bowl
import time
import pickle
import random
import os
import base64


class CONSTANTS:
    """
    Built in constants
    """
    PRIVATE_KEY_BASE_PATH = "private"
    PUBLIC_KEY_BASE_PATH = "public"
    ENCRYPTED_TEXT_FILE = "encrypted_text.txt"

    N = "n"
    P = "p"
    Q = "q"
    E = "e"
    D = "d"

    LOW_PRIMES = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61,
67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151,
157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241,
251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349,
353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443,
                449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523,
541, 547, 557, 563, 569, 571, 577, 587, 593, 599, 601, 607, 613, 617, 619, 631, 641,
643, 647, 653, 659, 661, 673, 677, 683, 691, 701, 709, 719, 727, 733, 739, 743, 751,
```

```
757, 761, 769, 773, 787, 797, 809, 811, 821, 823, 827, 829, 839, 853, 857, 859, 863,
877, 881, 883, 887, 907, 911, 919, 929, 937, 941, 947, 953, 967, 971, 977, 983, 991,
997]

   MILLER_RABIN_ITERATIONS = 128
   DEFAULT_KEY_SIZE = 50


class ERRORS:
   """
   Error messages
   """
   INVALID_CHOICE = "Please enter a valid Integer choice."
   INVALID_TEXT = "Please enter a valid Text. Text must only contain lowercase
alphabets."
   INVALID_KEY_IDENTITY = "Please enter a valid Key Identity. Key Identity must only
contain lowercase alphabets."
   DUPLICATE_KEY_FOUND = "Key with existing Key Identify found. Please enter a unique
Key Identity."


def miller_rabin_test(num, d):
   """
   Returns False if num is not prime, else True (likely prime)
   """
   a = random.randint(2, num-4)
   x = pow(a, int(d), num)

   # Fermat Little Theorem
   if x == 1 or x == num - 1:
       return True

   while d != num - 1:
       x = pow(x, 2, num)
       d *= 2
       if x == 1:
           return False
       if x == num-1:
           return True

   return False


def is_prime(num):
   """
   Returns True if number is likely to be prime, else False
```

```
    """

    if num < 2:
        return False

    if num in CONSTANTS.LOW_PRIMES:
        return True

    for prime in CONSTANTS.LOW_PRIMES:
        if num % prime == 0:
            return False

    c = num - 1
    while c % 2 == 0:
        c /= 2

    for _ in range(CONSTANTS.MILLER_RABIN_ITERATIONS):
        if not miller_rabin_test(num, c):
            return False

    return True


def egcd(a, b):
    """
    Calculates GCD using Euclid's Algorithm
    """
    s = 0
    old_s = 1
    t = 1
    old_t = 0
    r = b
    old_r = a

    while r != 0:
        quotient = old_r // r
        old_r, r = r, old_r - quotient * r
        old_s, s = s, old_s - quotient * s
        old_t, t = t, old_t - quotient * t

    # return gcd, x, y
    return old_r, old_s, old_t


def modular_inverse(a, b):
    """
```

```
    Calculates modular inverse of a with mod value b
    """
    _, x, _ = egcd(a, b)

    if x < 0:
        x += b

    return x


def is_coprime(num1, num2):
    """
    Returns True if GCD of num1 and num2 is 1, else False
    """
    return egcd(num1, num2)[0] == 1


def generate_large_prime(key_size=CONSTANTS.DEFAULT_KEY_SIZE):
    """
    Generates Large Prime Numbers with key_size bits
    """

    while True:
        num = random.randint(2**(key_size-1), 2**key_size - 1)

        if is_prime(num):
            return num


def choose_random_exponent(p, q, key_size=CONSTANTS.DEFAULT_KEY_SIZE):
    """
    Generates random exponent with key_size bits such that it is co-prime with
totient(p*q)
    """
    totient = (p-1)*(q-1)
    while True:
        gen = random.randint(2**(key_size-1), 2**key_size - 1)
        if is_coprime(gen, totient):
            return gen


def rsa_key_gen(text: str):
    """
    Generates RSA Public Key Private Key Pair identifying it with given text (usually
username)
    """
```

```
    text = text.lower()
    for c in text:
        if not c.isalpha():
            raise Exception(ERRORS.INVALID_KEY_IDENTITY)

    private_key_path = os.path.join(
        os.curdir, CONSTANTS.PRIVATE_KEY_BASE_PATH + "/"+text)
    public_key_path = os.path.join(
        os.curdir, CONSTANTS.PUBLIC_KEY_BASE_PATH + "/"+text)
    if os.path.isfile(private_key_path) or os.path.isfile(public_key_path):
        raise Exception(ERRORS.DUPLICATE_KEY_FOUND)

    p = generate_large_prime()
    q = generate_large_prime()
    e = choose_random_exponent(p, q)
    d = modular_inverse(e, (p-1)*(q-1))

    private_key = {CONSTANTS.N: p*q, CONSTANTS.P: p,
                   CONSTANTS.Q: q, CONSTANTS.E: e, CONSTANTS.D: d}
    public_key = {CONSTANTS.N:p*q,  CONSTANTS.E: e}

    pickle.dump(private_key, open(private_key_path, "wb"))
    pickle.dump(public_key, open(public_key_path, "wb"))

    print(f"Key with Identity {text} created successfully.")
    print(f"{CONSTANTS.N} : {p*q}")
    print(f"{CONSTANTS.P} : {p}")
    print(f"{CONSTANTS.Q} : {q}")
    print(f"{CONSTANTS.E} : {e}")
    print(f"{CONSTANTS.D} : {d}")


def rsa_cipher_encrypt(text: str, key_id: str):
    """
    RSA Cipher Encryption using RSA Public Key.
    Returns:
        Encrypted Text

    """
    key_file_path = os.path.join(
        os.curdir, CONSTANTS.PUBLIC_KEY_BASE_PATH+"/"+key_id)

    public_key = pickle.load(open(key_file_path, "rb"))

    cipher = ""
```

```
    for c in text:
        if (ord(c) < 97 or ord(c) > 122) and c != " ":
            raise Exception(ERRORS.INVALID_TEXT)


    for c in text:
        cipher += str(pow(ord(c), public_key[CONSTANTS.E], public_key[CONSTANTS.N])) +
" "
    cipher = cipher.strip()
    return base64.b64encode(cipher.encode("utf-8"))


def rsa_cipher_decrypt(key_id: str, file_path: str = CONSTANTS.ENCRYPTED_TEXT_FILE):
    """
    RSA Cipher Decryption using Private Key.
    Returns:
        Decrypted Text
    """
    with open(os.path.join(os.curdir, file_path), "r") as f:
        text = f.read()
    key_file_path = os.path.join(
        os.curdir, CONSTANTS.PRIVATE_KEY_BASE_PATH+"/"+key_id)

    private_key = pickle.load(open(key_file_path, "rb"))

    formatted_text = base64.b64decode(str.encode(text)).decode("utf-8")
    nums = [int(i) for i in formatted_text.split(" ")]
    decrypted_chars = []

    for num in nums:
        decrypted_chars.append(chr(pow(num, private_key[CONSTANTS.D],
private_key[CONSTANTS.N])))

    return "".join(decrypted_chars)


def rsa_key_gen_dialog():
    """
    Runs RSA Key Generation Dialog
    """
    text = input(
        "Please enter unique key identity (must be lowercase alphabets only): ")
    rsa_key_gen(text)


def rsa_cipher_encrypt_dialog():
```

```
    """
    Runs RSA Cipher Encryption Dialog
    """
    text = input("Enter text to be encrypted: ")
    key_identity = input("Enter key identity: ")

    encrypted_text = rsa_cipher_encrypt(text, key_identity)

    with open(os.path.join(os.curdir, CONSTANTS.ENCRYPTED_TEXT_FILE), "wb") as f:
        f.write(encrypted_text)

    print(f"Encrypted Text: {encrypted_text}")


def rsa_cipher_decrypt_dialog():
    """
    Runs RSA Cipher Decryption Dialog
    """
    key_identity = input("Enter key identity: ")

    decrypted_text = rsa_cipher_decrypt(key_identity)

    print(f"Decrypted Text: {decrypted_text}")


def main_dialog():
    """
    Runs main dialog sequence
    """
    try:
        choice = int(input(
            "RSA Cipher Program\n1. Encrypt\n2. Decrypt\n3. Generate Key Pair\nPlease
enter your choice: "))
    except Exception as e:
        raise Exception(ERRORS.INVALID_CHOICE)

    if choice == 1:
        rsa_cipher_encrypt_dialog()
    elif choice == 2:
        rsa_cipher_decrypt_dialog()
    elif choice == 3:
        rsa_key_gen_dialog()
    else:
        raise Exception(ERRORS.INVALID_CHOICE)
```

```python
if __name__ == "__main__":

    main_dialog()
```

```
kr@arc-warden:/mnt/6AD574E142A88B4D/BTech/Assignments/4th_Year/CNS/Assignment_8$ python3 1.py
RSA Cipher Program
1. Encrypt
2. Decrypt
3. Generate Key Pair
Please enter your choice: 3
Please enter unique key identity (must be lowercase alphabets only): svnit
Key with Identity svnit created successfully.
n : 401701792037068594714797565547
p : 688077079132603
q : 583803478156049
e : 648828616673227
d : 340761249739838239373170432 3
```

```
kr@arc-warden:/mnt/6AD574E142A88B4D/BTech/Assignments/4th_Year/CNS/Assignment_8$ python3 1.py
RSA Cipher Program
1. Encrypt
2. Decrypt
3. Generate Key Pair
Please enter your choice: 1
Enter text to be encrypted: once upon a time there was a little girl named goldilocks she went for a walk in the forest pretty
soon she came upon a house she knocked and when no one answered she walked right in at the table in the kitchen there were thre
e bowls of porridge goldilocks was hungry she tasted the porridge from the first bowl
Enter key identity: svnit
```

```
kr@arc-warden:/mnt/6AD574E142A88B4D/BTech/Assignments/4th_Year/CNS/Assignment_8$ python3 1.py
RSA Cipher Program
1. Encrypt
2. Decrypt
3. Generate Key Pair
Please enter your choice: 2
Enter key identity: svnit
Decrypted Text: once upon a time there was a little girl named goldilocks she went for a walk in the forest pretty soon she cam
e upon a house she knocked and when no one answered she walked right in at the table in the kitchen there were three bowls of p
orridge goldilocks was hungry she tasted the porridge from the first bowl
```