Lab4->project

```
import random

# (i) Setting. The alphabet will have 27 characters: the blank and the 26 letters of the English alphabet.
alphabet = "abcdefghijklmnopqrstuvwxyz "


# Check if a number is prime
def check_prime(n):
    if n < 2:
        return False
    if n > 2 and n % 2 == 0:
        return False
    for d in range(3, int(n ** 0.5) + 1, 2):  # Check odd divisors only
        if n % d == 0:
            return False
    return True


# (ii) Generate public and private keys
def generate_keys_rabin():
    while True:
        nr1 = random.randint(1000, 7000)
        nr2 = random.randint(1000, 7000)
        # nr1 and nr2 are prime and congruent to 3 (mod 4)
        if check_prime(nr1) and check_prime(nr2) and nr1 % 4 == 3 and nr2 % 4 == 3:
            break
    public_key = nr1 * nr2
    private_key = (nr1, nr2)
    return public_key, private_key


# (iii) Encrypt the plaintext
def encryption_rabin(plaintext, public_key):
    n = public_key
    encrypted_message = []
    for char in plaintext:
        # Validate the character
        if char not in alphabet:
            raise ValueError("Invalid character in plaintext.")
        m = alphabet.index(char)
        # Ciphertext c = m^2 mod n
        c = (m ** 2) % n
        encrypted_message.append(c)
    return encrypted_message


# (iv) Decrypt the ciphertext
def decrypt_rabin(ciphertext, private_key):
    p, q = private_key
    n = p * q
    decrypted_message = ""
    for c in ciphertext:
        # Validate ciphertext
        if not (0 <= c < n):
            raise ValueError("Invalid ciphertext.")

        # Compute modular square roots
        mp = pow(c, (p + 1) // 4, p)  # Square root mod p
        mq = pow(c, (q + 1) // 4, q)  # Square root mod q

        # Combine roots using CRT
        yp = pow(q, -1, p)  # q inverse mod p
        yq = pow(p, -1, q)  # p inverse mod q

        r1 = (mp * q * yp + mq * p * yq) % n
        r2 = (n - r1) % n
        r3 = (mp * q * yp - mq * p * yq) % n
        r4 = (n - r3) % n

        # Map roots to possible characters
        roots = [r1, r2, r3, r4]
        for root in roots:
            if root < len(alphabet):
                decrypted_message += alphabet[root]
                break
    return decrypted_message


# Main function to test the implementation
def main():
    public_key, private_key = generate_keys_rabin()
    print(f"Public Key: {public_key}")
    print(f"Private Key: {private_key}")

    plaintext = "example text"
    print(f"Original Plaintext: {plaintext}")
    encrypted_text = encryption_rabin(plaintext, public_key)
    print(f"Encrypted Text: {encrypted_text}")

    decrypted_text = decrypt_rabin(encrypted_text, private_key)
    print(f"Decrypted Text: {decrypted_text}")


# Run the main function
main()
```