**RSA – Confidentiality**

**import random**

**from math import gcd**

**def generate\_prime\_nos(bits):**

**while True:**

**num = random.getrandbits(bits)**

**if (num > 1) and all(num % d != 0 for d in range(2, int(num\*\*0.5) + 1)):**

**return num**

**def generate\_key\_pair(bits):**

**p = generate\_prime\_nos(bits // 2)**

**q = generate\_prime\_nos(bits // 2)**

**n = p \* q**

**phi = (p - 1) \* (q - 1)**

**e = random.randrange(2, phi)**

**while gcd(e, phi) != 1:**

**e = random.randrange(2, phi)**

**d = pow(e, -1, phi)**

**return ((e, n), (d, n))**

**def encrypt(alice\_public\_key, message):**

**e, n = alice\_public\_key**

**encrypted\_message = [pow(ord(char), e, n) for char in message]**

**return encrypted\_message**

**def decrypt(alice\_private\_key, encrypted\_message):**

**d, n = alice\_private\_key**

**decrypted\_message = ''.join(chr(pow(char, d, n)) for char in encrypted\_message)**

**return decrypted\_message**

**def main():**

**bits = int(input("Enter key size in bits (multiples of 8): "))**

**alice\_public\_key, alice\_private\_key = generate\_key\_pair(bits)**

**bob\_public\_key, bob\_private\_key = generate\_key\_pair(bits)**

**print("Alice's Public Key: ", alice\_public\_key)**

**print("Alice's Private Key: ", alice\_private\_key)**

**print("Bob's Public Key: ", bob\_public\_key)**

**print("Bob's Private Key: ", bob\_private\_key)**

**message = input("Enter the message for Bob to encrypt and send to Alice: ")**

**encrypted\_message = encrypt(alice\_public\_key, message)**

**print("Encrypted Message sent from Bob: ", encrypted\_message)**

**decrypted\_message = decrypt(alice\_private\_key, encrypted\_message)**

**print("Decrypted Message received by Alice: ", decrypted\_message)**

**if \_\_name\_\_ == "\_\_main\_\_":**

**main()**

**Digital Signature – Non Repudiation**

**import random**

**from math import gcd**

**def generate\_prime\_nos(bits):**

**while True:**

**num = random.getrandbits(bits)**

**if (num > 1) and all(num % d != 0 for d in range(2, int(num\*\*0.5) + 1)):**

**return num**

**def generate\_key\_pair(bits):**

**p = generate\_prime\_nos(bits // 2)**

**q = generate\_prime\_nos(bits // 2)**

**n = p \* q**

**phi = (p - 1) \* (q - 1)**

**e = random.randrange(2, phi)**

**while gcd(e, phi) != 1:**

**e = random.randrange(2, phi)**

**d = pow(e, -1, phi)**

**return ((e, n), (d, n))**

**def sign(alice\_private\_key, message):**

**d, n = alice\_private\_key**

**signed\_message = [pow(ord(char), d, n) for char in message]**

**return signed\_message**

**def verify(alice\_public\_key, signed\_message):**

**e, n = alice\_public\_key**

**verified\_message = ''.join(chr(pow(char, e, n)) for char in signed\_message)**

**return verified\_message**

**def main():**

**bits = int(input("Enter key size in bits (multiples of 8): "))**

**alice\_public\_key, alice\_private\_key = generate\_key\_pair(bits)**

**bob\_public\_key, bob\_private\_key = generate\_key\_pair(bits)**

**print("Alice's Public Key: ", alice\_public\_key)**

**print("Alice's Private Key: ", alice\_private\_key)**

**print("Bob's Public Key: ", bob\_public\_key)**

**print("Bob's Private Key: ", bob\_private\_key)**

**message = input("Enter the message for Alice to sign and send to Bob: ")**

**signature = sign(alice\_private\_key, message)**

**print("Message after being signed by Alice: ", signature)**

**verified\_message = verify(alice\_public\_key, signature)**

**if verified\_message == message:**

**print("Signature verified. The following message is sent by Alice:\n", verified\_message)**

**else:**

**print("Signature verification failed. Message is not sent by Alice.")**

**if \_\_name\_\_ == "\_\_main\_\_":**

**main()**

**RSA + Digital Signature (Confidentiality + Non Repudiation)**

**import random**

**from math import gcd**

**def generate\_prime\_nos(bits):**

**while True:**

**num = random.getrandbits(bits)**

**if (num > 1) and all(num % d != 0 for d in range(2, int(num\*\*0.5) + 1)):**

**return num**

**def generate\_key\_pair(bits):**

**p = generate\_prime\_nos(bits // 2)**

**q = generate\_prime\_nos(bits // 2)**

**n = p \* q**

**phi = (p - 1) \* (q - 1)**

**e = random.randrange(2, phi)**

**while gcd(e, phi) != 1:**

**e = random.randrange(2, phi)**

**d = pow(e, -1, phi)**

**return ((e, n), (d, n))**

**def encrypt(alice\_public\_key, message):**

**e, n = alice\_public\_key**

**encrypted\_message = [(pow(ord(char), e, n)) for char in message]**

**return encrypted\_message**

**def decrypt(private\_key, encrypted\_message):**

**d, n = private\_key**

**decrypted\_message = ''.join(chr(pow(char, d, n)) for char in encrypted\_message)**

**return decrypted\_message**

**def sign(bob\_private\_key, encrypted\_message):**

**d, n = bob\_private\_key**

**signed\_message = [(pow(char, d, n)) for char in encrypted\_message]**

**return signed\_message**

**def verify(bob\_public\_key, signed\_message):**

**e, n = bob\_public\_key**

**verified\_message = [(pow(char, e, n)) for char in signed\_message]**

**return verified\_message**

**def main():**

**bits = int(input("Enter key size in bits (multiples of 8): "))**

**alice\_public\_key, alice\_private\_key = generate\_key\_pair(bits)**

**bob\_public\_key, bob\_private\_key = generate\_key\_pair(bits)**

**print("Alice's Public Key: ", alice\_public\_key)**

**print("Alice's Private Key: ", alice\_private\_key)**

**print("Bob's Public Key: ", bob\_public\_key)**

**print("Bob's Private Key: ", bob\_private\_key)**

**message = input("Enter the message for Bob to encrypt and send to Alice: ")**

**encrypted\_message = encrypt(alice\_public\_key, message)**

**print("Encrypted Message sent from Bob: ", encrypted\_message)**

**signed\_message = sign(bob\_private\_key, encrypted\_message)**

**print("Message after being signed by Bob: ", signed\_message)**

**verified\_message = verify(bob\_public\_key, signed\_message)**

**if verified\_message == encrypted\_message:**

**print("Signature verified. The message is sent by Bob. ", verified\_message)**

**else:**

**print("Signature verification failed. Message is not sent by Bob.")**

**decrypted\_message = decrypt(alice\_private\_key, verified\_message)**

**print("Decrypted Message received by Alice: ", decrypted\_message)**

**if \_\_name\_\_ == "\_\_main\_\_":**

**main()**