RSA Public-Key Encryption and Signature Lab

SEED 2.0

Contents

Task 1

This code calculates the private key (d) based on the provided hexadecimal values for p, q, and e. It follows the RSA key generation algorithm to calculate the private exponent d using the given prime numbers p and q, and the public exponent e. The private key (d) is printed in hexadecimal form.

task1.py

```python
from Crypto.PublicKey import RSA

def calculate_private_key():
    p = int("F7E75FDC469067FFDC4E847C51F452DF", 16)
    q = int("E85CED54AF57E53E092113E62F436F4F", 16)
    e = int("0D88C3", 16)

    n = p * q
    phi_n = (p - 1) * (q - 1)

    # Calculate private key (d)
    d = pow(e, -1, phi_n)

    print("Private key (d):", hex(d))

if __name__ == "__main__":
    calculate_private_key()
```

Following is the calculated private key.

```
[09/30/23]seed@VM:~/.../RSA$ python3 task1.py
Private key (d): 0x3587a24598e5f2a21db007d89d18cc50aba5075ba19a33890fe7c28a9b496aeb
```

Task 2

This code demonstrates how to encrypt a message using the provided public key (n, e) in an RSA encryption scheme. It converts the message to hex, then to an integer, and encrypts it using the RSA encryption formula. The encrypted message (ciphertext) is printed in hexadecimal form.

task1.py | task2.py

```python
# Public key values
n = int("DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5", 16)
e = int("010001", 16)

# Message to encrypt
message = "A top secret!"

# Convert the message to hex and then to an integer
message_hex = int(message.encode("utf-8").hex(), 16)

# Encrypt the message
encrypted_message = pow(message_hex, e, n)

print("Encrypted message (C):", hex(encrypted_message))

```

Following is the encrypted message.

```
[09/30/23]seed@VM:~/.../RSA$ python3 task2.py
Encrypted message (C): 0x6fb078da550b2650832661e14f4f8d2cfaef475a0df3a75cacdc5de5cfc5fadc
```

Task 3

This code demonstrates how to decrypt a message using the provided public/private key (n, d) in an RSA encryption scheme. It uses the RSA decryption formula to decrypt the ciphertext and obtain the original message (plaintext). The decrypted message is then printed.

task1.py task2.py task3.py

```python
from Crypto.PublicKey import RSA

def decrypt_message():
    n = int("DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5", 16)
    d = int("74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D", 16)

    # Encrypted message (C)
    cipher = int("8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493F", 16)

    # Decrypt the message
    decrypted_message = pow(cipher, d, n)
    decrypted_message_bytes = decrypted_message.to_bytes((decrypted_message.bit_length() + 7) // 8, byteorder='big')

    print("Decrypted message:", decrypted_message_bytes.decode('utf-8'))

if __name__ == "__main__":
    decrypt_message()

```

Following is the decrypted message of the provided encrypted message.

```
[09/30/23]seed@VM:~/.../RSA$ python3 task3.py
Decrypted message: Password is dees
```

Task 4

This code demonstrates how to generate signatures for messages using the provided public/private key (n, d) in an RSA encryption scheme. It calculates the signature for both the original and a modified message by applying the RSA signing operation. The resulting signatures for both messages are printed in hexadecimal form.

task1.py | task2.py | task3.py | task4.py

```python
# Original message
M_original = "I owe you $2000."

# Provided private key (d) and public key (n)
d_hex = "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D"
n_hex = "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5"

d = int(d_hex, 16)
n = int(n_hex, 16)

# Sign the original message
M_original_hash = int(M_original.encode('utf-8').hex(), 16)
S_original = pow(M_original_hash, d, n)

# Print the original signature
print("Signature for the original message:", hex(S_original)[2:])

# Modify the message
M_modified = "I owe you $3000."

# Sign the modified message
M_modified_hash = int(M_modified.encode('utf-8').hex(), 16)
S_modified = pow(M_modified_hash, d, n)

# Print the modified signature
print("Signature for the modified message:", hex(S_modified)[2:])
```

It is clearly observable that only changing the value by only one character the signature has been changed. This tells that the signature is unique to the message provided.

```
[09/30/23]seed@VM:~/.../RSA$ python3 task4.py
Signature for the original message: 55a4e7f17f04ccfe2766e1eb32addba890bbe92a6fbe2d785ed6e73ccb35e4cb
Signature for the modified message: bcc20fb7568e5d48e434c387c06a6025e90d29d848af9c3ebac0135d99305822
```