Name: Abdurrahman Qureshi

Roll No: 242466

Practical No: 8

Date Of Performance: 01/09/2025

Aim: To implement the following algorithms:

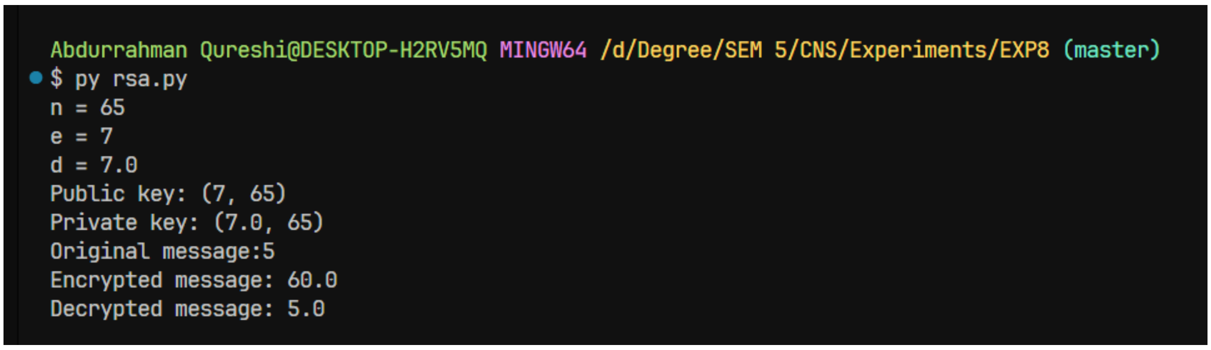
- RSA
- Digital Signature Scheme with RSA

RSA CODE:

```
import math

p = 5
q = 13

n = p*q
print("n =", n)

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

e = 7
while(e<phi):
    if (math.gcd(e, phi) == 1):
        break
    else:
        e += 1

print("e =", e)
k = 1
d = ((k*phi)+1)/e
```

```
print("d =", d)
print(f'Public key: {e, n}')
print(f'Private key: {d, n}')

msg = 5
print(f'Original message:{msg}')

C = pow(msg, e)
C = math.fmod(C, n)
print(f'Encrypted message: {C}')

M = pow(C, d)
M = math.fmod(M, n)

print(f'Decrypted message: {M}')
```

OUPTUT:



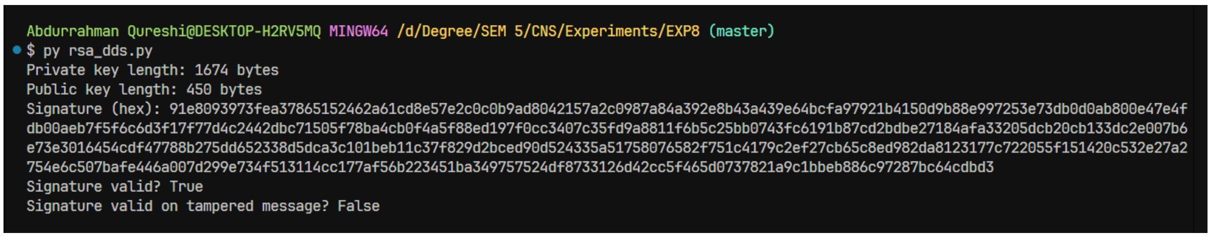
```
Abdurrahman Qureshi@DESKTOP-H2RV5MQ MINGW64 /d/Degree/SEM 5/CNS/Experiments/EXP8 (master)
$ py rsa.py
n = 65
e = 7
d = 7.0
Public key: (7, 65)
Private key: (7.0, 65)
Original message:5
Encrypted message: 60.0
Decrypted message: 5.0
```

Digital Signature Scheme with RSA CODE:

```
# rsa_sign_verify.py
from Crypto.PublicKey import RSA
from Crypto.Signature import pkcs1_15
from Crypto.Hash import SHA256

def generate_keys(bits=2048):
    key = RSA.generate(bits)
    private_pem = key.export_key()
    public_pem = key.publickey().export_key()
    return private_pem, public_pem

def sign_message(private_pem: bytes, message: bytes) -> bytes:
```

```python
    key = RSA.import_key(private_pem)
    h = SHA256.new(message)
    signer = pkcs1_15.new(key)
    signature = signer.sign(h)
    return signature

def verify_signature(public_pem: bytes, message: bytes, signature: bytes) -> bool:
    key = RSA.import_key(public_pem)
    h = SHA256.new(message)
    verifier = pkcs1_15.new(key)
    try:
        verifier.verify(h, signature)
        return True
    except (ValueError, TypeError):
        return False

if __name__ == "__main__":
    message = b"Hello! This is a message to sign with RSA."
    # Generate keys
    priv, pub = generate_keys(2048)
    print("Private key length:", len(priv), "bytes")
    print("Public key length:", len(pub), "bytes")

    # Sign
    sig = sign_message(priv, message)
    print("Signature (hex):", sig.hex())

    # Verify
    ok = verify_signature(pub, message, sig)
    print("Signature valid?", ok)

    # Tampering test
    tampered = b"Hello! This is a tampered message."
    ok2 = verify_signature(pub, tampered, sig)
    print("Signature valid on tampered message?", ok2)
```

OUTPUT:



```
Abdurrahman Qureshi@DESKTOP-H2RV5MQ MINGW64 /d/Degree/SEM 5/CNS/Experiments/EXP8 (master)
$ py rsa_dds.py
Private key length: 1674 bytes
Public key length: 450 bytes
Signature (hex): 91e8093973fea37865152462a61cd8e57e2c0c0b9ad8042157a2c0987a84a392e8b43a439e64bcfa97921b4150d9b88e997253e73db0d0ab800e47e4f
db00aeb7f5f6c6d3f17f77d4c2442dbc71505f78ba4cb0f4a5f88ed197f0cc3407c35fd9a8811f6b5c25bb0743fc6191b87cd2bdbe27184afa33205dcb20cb133dc2e007b6
e73e3016454cdf47788b275dd652338d5dca3c101beb11c37f829d2bced90d524335a51758076582f751c4179c2ef27cb65c8ed982da8123177c722055f151420c532e27a2
754e6c507bafe446a007d299e734f513114cc177af56b223451ba349757524df8733126d42cc5f465d0737821a9c1bbeb886c97287bc64cdbd3
Signature valid? True
Signature valid on tampered message? False
```

Performance (7M)	Journal (3M)	Lab Ethics (2M)	Attendance (3M)	Total (15M)	Faculty Signature