RSA

```
import math

def gcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a

def mod_inverse(e, phi):
    for d in range(1, phi):
        if (e * d) % phi == 1:
            return d
    return None

def rsa_demo(p, q, message):
    n = p * q
    phi = (p - 1) * (q - 1)
    e = 2
    while e < phi:
        if gcd(e, phi) == 1:
            break
        e += 1
    d = mod_inverse(e, phi)
    cipher = pow(message, e, n)
    decrypted = pow(cipher, d, n)
    return {"Public Key": (e, n), "Private Key": (d, n), "Cipher": cipher, "Decrypted": decrypted}

print("Example 1:", rsa_demo(17, 11, 88))
print("Example 2:", rsa_demo(13, 7, 9))
```

Diffie Hellman

```
def diffie_hellman(p, g, a, b):
    A = pow(g, a, p)
    B = pow(g, b, p)
    secret_A = pow(B, a, p)
    secret_B = pow(A, b, p)
    print(f"Prime (p): {p}")
    print(f"Base (g): {g}")
    print(f"Alice Private Key: {a}")
    print(f"Bob Private Key: {b}")
    print(f"Alice Public Key: {A}")
    print(f"Bob Public Key: {B}")
    print(f"Shared Secret (Alice): {secret_A}")
    print(f"Shared Secret (Bob): {secret_B}")
    print("-" * 40)
```

```
print("Example 1:")
diffie_hellman(23, 5, 6, 15)

print("Example 2:")
diffie_hellman(17, 3, 4, 9)
```

SHA - 256

```
def right_rotate(value, bits):
    return ((value >> bits) | (value << (32 - bits))) & 0xFFFFFFFF

def sha256(message):
    h = [
        0x6a09e667, 0xbb67ae85, 0x3c6ef372, 0xa54ff53a,
        0x510e527f, 0x9b05688c, 0x1f83d9ab, 0x5be0cd19
    ]
    k = [
        0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5, 0x3956c25b, 0x59f111f1,
        0x923f82a4, 0xab1c5ed5, 0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3,
        0x72be5d74, 0x80deb1fe, 0x9bdc06a7, 0xc19bf174, 0xe49b69c1, 0xefbe4786,
        0x0fc19dc6, 0x240ca1cc, 0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, 0x76f988da,
        0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7, 0xc6e00bf3, 0xd5a79147,
        0x06ca6351, 0x14292967, 0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13,
        0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85, 0xa2bfe8a1, 0xa81a664b,
        0xc24b8b70, 0xc76c51a3, 0xd192e819, 0xd6990624, 0xf40e3585, 0x106aa070,
        0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5, 0x391c0cb3, 0x4ed8aa4a,
        0x5b9cca4f, 0x682e6ff3, 0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208,
        0x90befffa, 0xa4506ceb, 0xbef9a3f7, 0xc67178f2
    ]
    message_bytes = bytearray(message, 'utf-8')
    orig_len_in_bits = (8 * len(message_bytes)) & 0xffffffffffffffff
    message_bytes.append(0x80)
    while (len(message_bytes) * 8) % 512 != 448:
        message_bytes.append(0)
    message_bytes += orig_len_in_bits.to_bytes(8, 'big')
    for chunk_start in range(0, len(message_bytes), 64):
        chunk = message_bytes[chunk_start:chunk_start + 64]
        w = [int.from_bytes(chunk[i:i+4], 'big') for i in range(0, 64, 4)]
        for i in range(16, 64):
            s0 = right_rotate(w[i-15], 7) ^ right_rotate(w[i-15], 18) ^ (w[i-15] >> 3)
            s1 = right_rotate(w[i-2], 17) ^ right_rotate(w[i-2], 19) ^ (w[i-2] >> 10)
            w.append((w[i-16] + s0 + w[i-7] + s1) & 0xFFFFFFFF)
        a, b, c, d, e, f, g, h_temp = h
        for i in range(64):
            S1 = right_rotate(e, 6) ^ right_rotate(e, 11) ^ right_rotate(e, 25)
            ch = (e & f) ^ ((~e) & g)
            temp1 = (h_temp + S1 + ch + k[i] + w[i]) & 0xFFFFFFFF
            S0 = right_rotate(a, 2) ^ right_rotate(a, 13) ^ right_rotate(a, 22)
```

```
            maj = (a & b) ^ (a & c) ^ (b & c)
            temp2 = (S0 + maj) & 0xFFFFFFFF
            h_temp = g
            g = f
            f = e
            e = (d + temp1) & 0xFFFFFFFF
            d = c
            c = b
            b = a
            a = (temp1 + temp2) & 0xFFFFFFFF
        h = [(x + y) & 0xFFFFFFFF for x, y in zip(h, [a, b, c, d, e, f, g, h_temp])]
    return ''.join(format(x, '08x') for x in h)

message = "Hello World"
print("Input:", message)
print("SHA-256 Hash:", sha256(message))
```

Digital Signature

```
import random

def mod_inverse(a, m):
    return pow(a, -1, m)

def hash_message(msg, q):
    return sum((ord(c) * (i + 1) for i, c in enumerate(msg))) % q

def dss_sign(message, p, q, g, private_key):
    h = hash_message(message, q)
    while True:
        k = random.randint(1, q - 1)
        r = pow(g, k, p) % q
        if r == 0:
            continue
        k_inv = mod_inverse(k, q)
        s = (k_inv * (h + private_key * r)) % q
        if s == 0:
            continue
        return s, r

def dss_verify(message, s, r, p, q, g, public_key):
    if not (0 < r < q and 0 < s < q):
        return False
    h = hash_message(message, q)
    w = mod_inverse(s, q)
    u1 = (h * w) % q
    u2 = (r * w) % q
    v = (pow(g, u1, p) * pow(public_key, u2, p) % p) % q
```

```
    return v == r

if __name__ == "__main__":
    q = 11
    p = 23
    g = 4
    private_key = 6
    public_key = pow(g, private_key, p)

    message = "network"
    s, r = dss_sign(message, p, q, g, private_key)

    print(f"Message: {message}")
    print(f"Signature (s, r): ({s}, {r})")

    valid = dss_verify(message, s, r, p, q, g, public_key)
    if valid:
        print("Signature is VERIFIED for the original message.")
    else:
        print("Signature is NOT VERIFIED for the original message.")

    tampered = "worknet"
    print(f"Tampered Message: {tampered}")

    valid_tampered = dss_verify(tampered, s, r, p, q, g, public_key)
    if valid_tampered:
        print("Signature is VERIFIED for the tampered message. (Error!)")
    else:
        print("Signature is NOT VERIFIED for the tampered message.")
```

Secure Socket Layer (SSL)

Steps with Explanations

Generate RSA Private Key

```
openssl genrsa -out private.key 2048
```

1. 🔹 *Generates a 2048-bit RSA private key, used for encryption and signing.*

Generate Certificate Signing Request (CSR)

```
openssl req -new -key private.key -out request.csr
```

2. 🔹 *Creates a CSR containing public key and organization info, digitally signed with the private key.*

Self-sign the CSR to Create a Certificate

```
openssl x509 -req -days 365 -in request.csr -signkey private.key
-out certificate.crt
```

3. 🔹 *Generates a self-signed digital certificate valid for 1 year. This certificate will be used in SSL communication.*

Verify and Inspect the CSR

```
openssl req -text -noout -verify -in request.csr
```

4. 🔹 *Displays and verifies the CSR to ensure it's valid and correctly formatted.*

Set Up CA Directory Structure

```
mkdir -p demoCA/newcerts
touch demoCA/index.txt
echo 1000 > demoCA/serial
```

5. 🔹 *Prepares a minimal Certificate Authority (CA) structure for managing certificates.*

Revoke the Certificate

```
openssl ca -revoke certificate.crt -keyfile private.key -cert
certificate.crt
```

6. 🔹 *Revokes a certificate if it is compromised or expired.*

Cleanup Temporary Files

```
rm request.csr
```

7. 🔹 *Removes unnecessary CSR file to maintain security of private information.*

Intrusion Detection System (IDS)

Steps with Explanations

Install Snort

```
sudo apt update
sudo apt install snort
```

1. 🔹 *Updates system packages and installs Snort IDS.*

Check Network Interfaces

```
ip link show
```

2. 🔹 *Displays all network interfaces (like lo, eth0, wlan0). You'll choose one for Snort monitoring.*

Edit Snort Configuration

```
sudo nano /etc/snort/snort.conf
```

3. 🔹 *Configures Snort settings, ensuring `$RULE_PATH` points to `/etc/snort/rules` and correct interfaces are used.*

Add Local Rules

```
sudo nano /etc/snort/rules/local.rules
```

Example Rules:

```
alert tcp any any -> any any (msg:"Possible attack detected";
content:"attack"; nocase; sid:1000001; rev:1;)
alert udp any any -> any any (msg:"Possible attack detected in UDP";
content:"attack"; nocase; sid:1000002; rev:1;)
alert ip any any -> any any (msg:"Possible attack detected in IP";
content:"attack"; nocase; sid:1000003; rev:1;)
```

4. 🔹 *Defines custom Snort rules to trigger alerts when packets contain specific keywords like "attack".*

Test Snort Configuration

```
sudo snort -c /etc/snort/snort.conf -T
```

5. 🔹 *Checks Snort configuration for syntax errors before running.*

Run Snort in Console Mode

```
sudo snort -A console -q -c /etc/snort/snort.conf -i <interface>
```

6. 🔹 ***Starts Snort in live monitoring mode to detect attacks in real-time.***

Send Test UDP Traffic

```
echo "this is an attack" | nc -u -w1 127.0.0.1 12345
```

7. 🔹 ***Sends a UDP packet with the keyword "attack" to test Snort's detection rules.***

8. **Verify Snort Alerts**
   🔹 ***Check Snort's console or log files to ensure an alert was generated, confirming IDS functionality.***