RSA

```
import math
p=int(input('ENter prime number p'))
q=int(input('Enter prime number q'))
n=p*q
print('n:',n)
phi=(p-1)*(q-1)
for i in range(2,phi):
    if math.gcd(i,phi)==1:
        e=i
        break
d=pow(e,-1,phi)
msg=int(input('Enter msg m<n'))
if msg>n:
    print('msg must be less than n')
    exit()
ct=pow(msg,e,n)
dt=pow(ct,d,n)
print("Public key:", (e, n))
print("Private key:", (d, n))
print("Encrypted message:", ct)
print("Decrypted message:", pow(ct, d, n))
```

Diffie-Hellman

```
import random
p = int(input("Enter prime number p:"))
q = int(input("Enter primitive root q:"))
a = random.randint(2, p - 2)
A = pow(q, a, p)
print(f"\n[Alice] Private Key (a): {a}")
print(f"[Alice] Public Key (A): {A}")
b = random.randint(2, p - 1)
B = pow(q, b, p)
print(f"\n[Bob] Private Key (b): {b}")
print(f"[Bob] Public Key (B): {B}")
shared_secret_alice = pow(B, a, p)
shared_secret_bob = pow(A, b, p)
print(f"\n[Alice] Computed Shared Secret: {shared_secret_alice}")
print(f"[Bob] Computed Shared Secret:   {shared_secret_bob}")
if shared_secret_alice == shared_secret_bob:
    print("\n Shared secret successfully established!")
else:
    print("\n Shared secret mismatch! Something went wrong.")
```

DSS

```python
import random
def mod_inverse(a, m):
    return pow(a, -1, m)
def hash_msg(msg, q):
    return sum(ord(c)*(i+1) for i, c in enumerate(msg)) % q
def sign(msg, p, q, g, x):
    h = hash_msg(msg, q)
    while True:
        k = random.randint(1, q-1)
        r = pow(g, k, p) % q
        if not r:
            continue
        s = (mod_inverse(k, q) * (h + x*r)) % q
        if s:
            return s, r
def verify(msg, s, r, p, q, g, y):
    if not (0 < r < q and 0 < s < q):
        return False
    h= hash_msg(msg, q)
    w=mod_inverse(s, q)
    v = (pow(g, h*w % q, p) * pow(y, r*w % q, p) % p) % q
    return v == r
p, q, g = 23, 11, 4
x = 6                               # private key
y = pow(g, x, p)                    # public key
msg = input("Enter a msg:")
s, r = sign(msg, p, q, g, x)
print(f"Message: {msg}\nSignature: (s={s}, r={r})")
print("Verified:", verify(msg, s, r, p, q, g, y))
tampered = input("Enter a tampered msg:")
print(f"\nTampered: {tampered}")
print("Verified:", verify(tampered, s, r, p, q, g, y))
```

OPENSSL

sudo apt remove openssl

sudo apt-get install openssl

openssl version

1Create Folder Structure

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

→ Prepares CA working directories and database files.

2Generate CA Private Key

```
openssl genrsa -out demoCA/private/cakey.pem 2048
```

→ Creates a 2048-bit private key for the Certificate Authority.

3Create Self-Signed CA Certificate

```
openssl req -x509 -new -key demoCA/private/cakey.pem -out demoCA/cacert.pem -
days 365
```

→ Generates a CA certificate valid for 1 year.

4Generate Server Private Key

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

→ Generates a private key for the server.

5 Create Certificate Signing Request (CSR)

```
openssl req -new -key server.key -out server.csr
```

→ Creates a CSR containing server information.

6 Sign CSR with CA (Issue Certificate)

```
openssl x509 -req -in server.csr -CA demoCA/cacert.pem -CAkey
demoCA/private/cakey.pem -CAcreateserial -out server.crt -days 365
```

→ Issues a signed certificate for the server using the CA.

7 Verify Certificate

```
openssl verify -CAfile demoCA/cacert.pem server.crt
```

→ Checks that the server certificate is valid and signed by the CA.

8Revoke Certificate

```
openssl ca -revoke server.crt -keyfile demoCA/private/cakey.pem -cert
demoCA/cacert.pem
```

→ Revokes the certificate and updates CA’s index file.

Snort

Sudo apt install snort

1) Ensure the real rules directory exists (create if missing)

```
sudo mkdir -p /etc/snort/rules
sudo touch /etc/snort/rules/local.rules
```

→ Create system rules dir and an empty `local.rules` (needs sudo).

---

2) Edit `/etc/snort/rules/local.rules` and add your custom rules

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

Paste for example (3 variants):

```
alert tcp any any -> any any (msg:"Possible attack detected (TCP)";
content:"attack"; nocase; sid:1000001; rev:1;)
alert udp any any -> any 12345 (msg:"Possible attack detected (UDP)";
content:"attack"; nocase; sid:1000002; rev:1;)
alert ip  any any -> any any (msg:"Possible attack detected in any IP";
content:"attack"; nocase; sid:1000003; rev:1;)
```

Save & exit.

---

3) Make sure `snort.conf` includes the local rules and correct `$RULE_PATH`

Quick check (show lines):

```
sudo grep -E "RULE_PATH|local.rules" /etc/snort/snort.conf -n
```

If `include $RULE_PATH/local.rules` is missing, add it inside `snort.conf` (edit with sudo nano).

→ Ensures Snort will load your `local.rules`.

---

4) Test/validate the configuration (always do this before running)

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

→ Syntax & rule parsing test. Expect: `Snort successfully validated the configuration!`

If it fails, read the error and fix the referenced file/path.

---

5) Start Snort in foreground (recommended for lab testing) — Terminal A

Do this in one terminal and leave it running:

```
sudo snort -A console -q -c /etc/snort/snort.conf -i lo
```

→ Runs Snort on loopback `lo`, prints alerts to console. Don't suspend with Ctrl+Z.

---

6) Send test packet in Terminal B (to trigger UDP rule on port 12345)

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

→ Netcat sends a UDP packet containing `attack` to loopback port 12345.