Programming Task (Using SageMath)

RSA Implementation

a) Generate random primes p and q, each 1000+ bits in size.
b) Use the public key e = 65537.
c) Generate the private key d.
d) Generate a random 256-bit AES key (treat this as the message m).
e) Encrypt m using the public key: Compute ciphertext c = m^e mod n (where n = p*q).
f) Hash the message m: Input m into SHA-256, producing a fingerprint f1.
g) Sign f1 with the private key d: Compute digital signature DS = f1^d mod n.

1. Key Generation → **2. Encryption** → **3. Decryption** → **4. Hashing & Signature**

1. Key Generation

- Generate 1024-bit primes **p** and **q**
- Compute **n = p × q** and **ϕ(n) = (p-1)(q-1)**
- Set public key **e = 65537** (verify **gcd(ϕ(n),e) = 1**)
- Compute private key **d =** e^{-1} **mod ϕ(n)**

2. Encryption

- Generate 256-bit random **AES key (m)**
- Encrypt: **c =** m^e **mod n**

3. Decryption

- Decrypt: **m =** c^d **mod n**
- Verify **decrypted m == original m**

4. Hashing & Signature

- Hash **m** with SHA-256 → **f1**
- Sign: **DS =** $f1^d$ **mod n**

Programming Task (Using SageMath)

Diffie-Hellman (DH) Key Exchange

- Perform a DH calculation similar to the RSA example above.
- Use 1000+ bit parameters (e.g., safe primes). random primes p and q, each 1000+ bits in size.

1. Parameter Setup

- **Generate** 1000+ bit prime **q**
- **Find** primitive root **g** of q

2. Private Key Generation

- Alice: Select private key **a** ($1 < a < q-1$)
- Bob: Select private key **b** ($1 < b < q-1$)

3. Public Key Exchange

- Alice computes: $\mathbf{A = g^a \bmod q}$
- Bob computes: $\mathbf{B = g^b \bmod q}$

4. Shared Secret Calculation

- Alice computes: $\mathbf{s = B^a \bmod q}$
- Bob computes: $\mathbf{s = A^b \bmod q}$
- **Verify**: s_Alice == s_Bob

Programming Task (Using SageMath)

ECDSA (Elliptic Curve Digital Signature Algorithm)

- Perform an ECDSA calculation similar to the RSA example.
- Use ECC with at least 256-bit curves

NOTES:

1. **SageMath Reference**: Elliptic curve operations: SageMath Finite Field Elliptic Curves Documentation
2. **Curve Parameters**:
   - Prime modulus (**q**): From SafeCurves Project
   - Coefficients (**a**, **b**): Verified via SafeCurves Equation Guide to satisfy the elliptic curve equation: $y^2 = x^3 + ax + b$
   - Base point (**G**): Coordinates from SafeCurves Base Points
3. **Hash Assumption**: Message hash (**e = 13**): Simplified value for demonstration (based on slide 22 of "Digital Signature" lecture notes).

1. Parameter Setup

- **Prime (q)**: Predefined 256-bit prime
- **Curve Coefficients**: a = -3, b (predefined)
- **Base Point (G)**: Predefined (x,y) coordinates
- **Order (n)**: Order of G

2. Key Generation

- **Private Key (d)**: Random integer ∈ [1, n-1]
- **Public Key (Q)**: Compute Q = d × G

3. Signature Generation

- Generate random **k** ∈ [1, n-1]
- Compute P = k × G → r = x_P mod n (repeat if r=0)
- Compute hash **e = SHA-256(m)** (simplified as e=13 here)
- Calculate s = k^{-1} × (e + d×r) mod n (repeat if s=0)
- Signature: Pair (r, s)

4. Signature Verification

- Compute w = s^{-1} mod n
- Calculate u1 = e×w mod n, u2 = r×w mod n
- Compute X = u1×G + u2×Q
- **Verify**: v = x_X mod n matches r