BARI ANKIT (56)

Exp 2 : Implementation and analysis of RSA cryptosystem and Digital signature scheme using RSA/El Gamal.

Implementation and Analysis of RSA Cryptosystem and Digital Signature Scheme

Overview:

1. **RSA Cryptosystem:**
    - **The RSA algorithm is a widely-used asymmetric encryption technique. It relies on the mathematical properties of large prime numbers and their difficulty to factorize. It involves two keys: a public key (used for encryption) and a private key (used for decryption).**
    - **The security of RSA is based on the difficulty of factoring large composite numbers.**
2. **Digital Signature Scheme:**
    - **A digital signature is a cryptographic technique used to verify the authenticity and integrity of digital messages or documents.**
    - **In RSA, the digital signature is created by encrypting the hash of the message with the private key and can be verified using the public key.**
    - **ElGamal is another popular public-key cryptosystem, and it can also be used for digital signatures.**

In this answer, we'll walk through both the RSA cryptosystem and the Digital Signature Scheme using RSA and ElGamal.

1. RSA Cryptosystem:

Key Generation:

- **Choose two large prime numbers, p and q.**
- **Compute n=p×qn = p \times qn=p×q. The number nnn is used as the modulus for both the public and private keys.**
- **Compute the totient function ϕ(n)=(p−1)(q−1)\phi(n) = (p - 1)(q - 1)ϕ(n)=(p−1)(q−1).**
- **Choose an encryption exponent eee such that 1<e<ϕ(n)1 < e < \phi(n)1<e<ϕ(n) and gcd(e,ϕ(n))=1\gcd(e, \phi(n)) = 1gcd(e,ϕ(n))=1.**
- **Compute the decryption exponent ddd such that d×e≡1 (mod ϕ(n))d \times e \equiv 1 \,(\text{mod}\, \phi(n))d×e≡1(modϕ(n)).**
- **The public key is (e,n)(e, n)(e,n), and the private key is (d,n)(d, n)(d,n).**

Encryption:

- **The sender encrypts the plaintext message MMM using the public key** (e,n)(e, n)(e,n) by computing: C=Me (mod n)C = M^e \,(\text{mod} \, n)C=Me(modn) where MMM is the message, eee is the public exponent, and nnn is the modulus.

Decryption:

- The receiver decrypts the ciphertext CCC using the private key (d,n)(d, n)(d,n) by computing: M=Cd (mod n)M = C^d \,(\text{mod} \, n)M=Cd(modn) where CCC is the ciphertext, ddd is the private exponent, and nnn is the modulus.

2. Digital Signature Scheme using RSA:

- A digital signature is a mathematical scheme that allows a sender to sign a message so that the receiver can verify both the authenticity and integrity of the message.

Steps to Create a Digital Signature using RSA:

1. Hash the Message: First, compute a hash of the message MMM using a cryptographic hash function (e.g., SHA-256).
2. Sign the Hash: The sender signs the hash H(M)H(M)H(M) by encrypting it with their private key:

Signature=H(M)d (mod n)\text{Signature} = H(M)^d \,(\text{mod} \, n)Signature=H(M)d(modn)

3. Send the Signature: The sender sends both the message and the digital signature to the receiver.

Verification of Digital Signature using RSA:

1. The receiver computes the hash H(M)H(M)H(M) of the received message MMM.
2. The receiver decrypts the signature using the public key of the sender to obtain:

Decrypted signature=Signaturee (mod n)\text{Decrypted signature} = \text{Signature}^e \,(\text{mod} \, n)Decrypted signature=Signaturee(modn)

3. The receiver checks if the decrypted signature matches the hash of the message:
   - If the decrypted signature equals H(M)H(M)H(M), the signature is valid.
   - If they do not match, the signature is invalid, and the message has been tampered with.

3. ElGamal Digital Signature Scheme:

ElGamal is another public-key cryptosystem based on the Diffie-Hellman key exchange. It can also be used for digital signatures. The main steps in the ElGamal Digital Signature Scheme are:

Key Generation for ElGamal:

1. Choose a large prime ppp and a primitive root ggg modulo ppp.
2. Choose a private key xxx where 1<x<p−11 < x < p-11<x<p−1.

3. **Compute the corresponding public key y=gx (mod p)y = g^x \,(\text{mod} \, p)y=gx(modp).**

Signing Process:

1. **Compute a cryptographic hash of the message H(M)H(M)H(M).**
2. **Choose a random value kkk such that 1<k<p−11 < k < p-11<k<p−1 and gcd(k,p−1)=1\gcd(k, p-1) = 1gcd(k,p−1)=1.**
3. **Compute the signature pair (r,s)(r, s)(r,s): r=gk (mod p)r = g^k \,(\text{mod} \, p)r=gk(modp) s=k−1×(H(M)−x×r) (mod p−1)s = k^{-1} \times (H(M) - x \times r) \,(\text{mod} \, p-1)s=k−1×(H(M)−x×r)(modp−1)**

Verification Process:

1. **Compute the hash H(M)H(M)H(M) of the received message.**
2. **Compute the values v1v_1v1 and v2v_2v2 using the public key and signature: v1=yr×rs (mod p)v_1 = y^r \times r^s \,(\text{mod} \, p)v1 =yr×rs(modp) v2=gH(M) (mod p)v_2 = g^{H(M)} \,(\text{mod} \, p)v2 =gH(M)(modp)**
3. **If v1=v2v_1 = v_2v1=v2, the signature is valid.**

Code :

```
import random
from sympy import isprime

# Function to find GCD
def gcd(a, b):
    while b:
        a, b = b, a % b
    return a

# Function to find modular inverse
def mod_inverse(a, m):
    m0, x0, x1 = m, 0, 1
    while a > 1:
        q = a // m
        m, a = a % m, m
        x0, x1 = x1 - q * x0, x0
    if x1 < 0:
        x1 += m0
```

```
    return x1

# RSA Key Generation
def rsa_keygen(bitsize=512):
    p = q = 0
    while not isprime(p):
        p = random.getrandbits(bitsize)
        if isprime(p): break
    while not isprime(q) or p == q:
        q = random.getrandbits(bitsize)
        if isprime(q): break
    n = p * q
    phi_n = (p - 1) * (q - 1)

    e = 65537  # common choice for public exponent
    d = mod_inverse(e, phi_n)

    return (e, n), (d, n)

# RSA Encryption
def rsa_encrypt(message, pub_key):
    e, n = pub_key
    return [pow(ord(char), e, n) for char in message]

# RSA Decryption
def rsa_decrypt(ciphertext, priv_key):
    d, n = priv_key
    return ''.join([chr(pow(char, d, n)) for char in ciphertext])

# Digital Signature (RSA)
def rsa_sign(message, priv_key):
    d, n = priv_key
    message_hash = hash(message)
    signature = pow(message_hash, d, n)
```

```python
    return signature

# RSA Signature Verification
def rsa_verify(message, signature, pub_key):
    e, n = pub_key
    message_hash = hash(message)
    decrypted_signature = pow(signature, e, n)
    return decrypted_signature == message_hash
```

Output:

```python
public_key, private_key = rsa_keygen()

message = "Hello, RSA!"
ciphertext = rsa_encrypt(message, public_key)
print(f"Encrypted message: {ciphertext}")

decrypted_message = rsa_decrypt(ciphertext, private_key)
print(f"Decrypted message: {decrypted_message}")

# Digital Signature
signature = rsa_sign(message, private_key)
print(f"Signature: {signature}")

# Verify Signature
is_valid = rsa_verify(message, signature, public_key)
print(f"Signature valid: {is_valid}")
```