Cyber Security

Lab Assignment- 7(A)

Objective: Implement the RSA encryption system.

Note: You can use the 'Integer' , 'PrimeAndGenerator' and the 'AutoSeededRandomPool ' classes , do not use any other libraries function from Crypto++.

Process phase implementation.

1. Setup Phase
2. Encryption Phase
3. Decryption Phase

1. Setup Phase:

Objective:

1. Generate the two large prime numbers p and q, such that ($q \neq p$).

   Computation phase:
   1. Calculate the n: $n = p \times q$
   2. Calculate the $\Phi(n)$: $\Phi(n) = (p-1) \times (q-1)$

2. Generate private key (d):

$$d \xleftarrow{R} Z_{\Phi(n)}^*$$

3. Calculate the public key (e):

$$e \equiv d^{-1} \bmod n$$

4. Store the generated public key parameters (e, n) in the binary file named **'publickey.bin'** .

5. Store the generated private key in separate binary file '**privatekey.bin**'.

Note: 1. d is coprime to Φ(n).

2. Discard the value of p, q, Φ(n) after the key generation.

2. **Encryption Phase:**

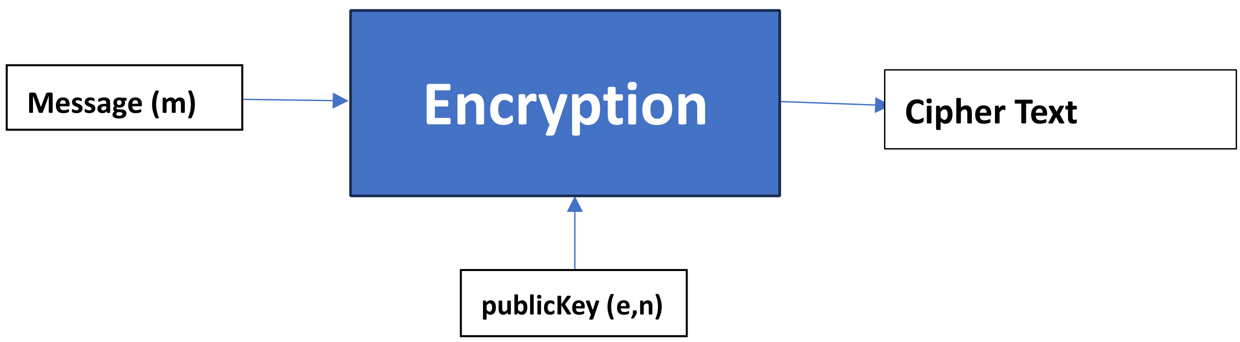
Objective: To generate the Cipher text from the Plain text using the public key (e, n) .

Computation Process:

1. Input the plain text file (m) path in the user terminal.
2. To generate the Ciphertext (C)

$$C \equiv m^e \bmod n$$

Workflow:



3. **Decryption Phase:**

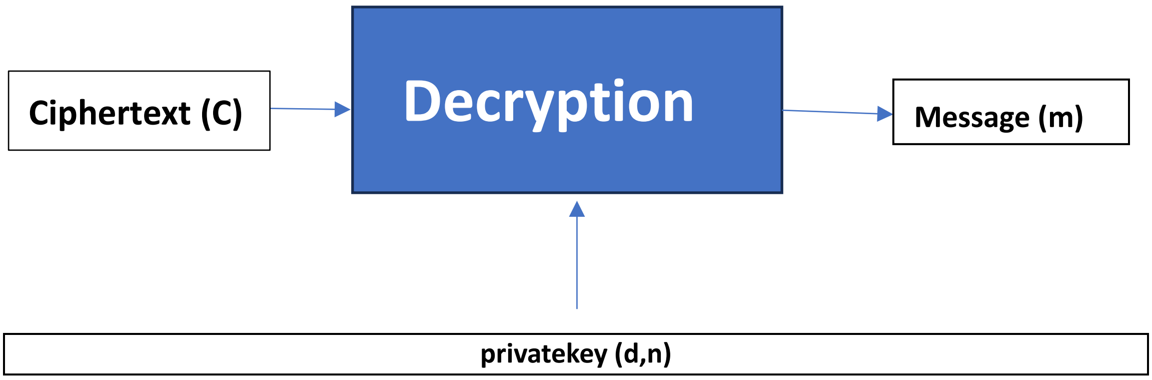
Objective: To generate the plain text from the Ciphertext (C) using the private key (d, n).

Computation Process:

Inputs: Ciphertext: C , Private key: (d, n) **Outputs:** Plaintext:

$$m \equiv C^d \bmod n$$

Workflow:



Lab Assignment- 7(B)

Objective: Implement RSA Digital Signature System.

Note: You can use the 'Integer' , 'PrimeAndGenerator' and the 'AutoSeededRandomPool ' classes , do not use any other libraries function from Crypto++.

Process phase implementation.

1. Setup Phase
2. Signature Phase
3. Verification Phase

1. **Setup Phase:**
   Objective:
2. Generate the two large prime numbers p and q, such that ($q \neq p$).

Computation phase:

3. Calculate the n: $n = p \times q$

4. Calculate the $\Phi(n)$:
$\Phi(n) = (p-1) \times (q-1)$

2. Generate private key (d):

$$d \xleftarrow{R} Z_{\Phi(n)^*}$$

3. Calculate the public key (e):

$$e \equiv d^{-1} \bmod n$$

4. Store the generated public key parameters (e, n) in the binary file named **'publickey.bin'** .

5. Store the generated private key in separate binary file '**privatekey.bin**'.

Note: 1. d is coprime to $\Phi(n)$.
2. Discard the value of p, q, $\Phi(n)$ after the key generation.

2. Signature Phase:

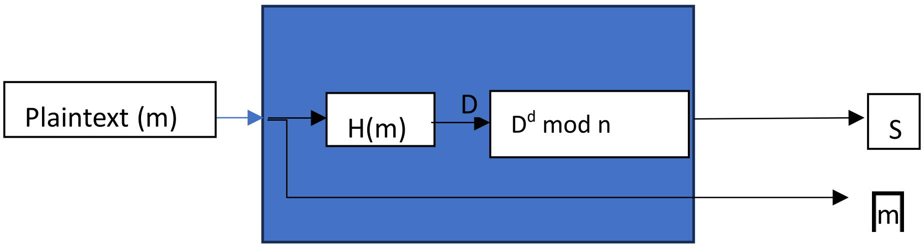
Objective: 1. Generate the hash of the plaintext (m) using the md5sum. 2. Store the generated hash in the binary file named '**msgHash1.bin**'.

3. Compute the signed hashed message (H(m)) using the private key (d) .

Computation:

1. To find the hash refer to the assignment 6.
2. Signed the generated hash message. $S \equiv (H(m))^d \bmod n$

Workflow:



3. **Verification Phase:**

Objective: 1. To verify the signature using the public key (e), by following the computations.

Computations:

1. Verify the signature using the private key (e).

   $D' \equiv (S)^e \bmod n$

2. Use the message hash file '**msgHash1.bin**' (D) to compare with the hash D'.

   D=D'

Note: Store the value of the D' in separate binary file named '**msgHash2.bin**'.

Workflow: