Information Security Lab 5

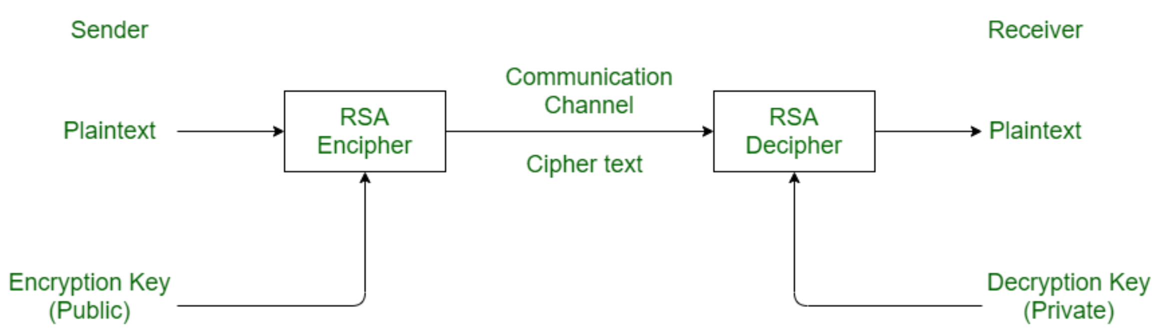
Aim: To Implement RSA and DSA Algorithm

Name: - Shri Darandale

PRN: - 20210801066

1. Rivest-Shamir-Adleman (RSA) algorithm :

RSA stands for **Rivest-Shamir-Adleman**. It is a cryptosystem used for secure data transmission. In RSA algorithm, encryption key is public but decryption key is private. This algorithm is based on mathematical fact that factoring the product of two large prime numbers is not easy. It was developed by **Ron Rivest**, **Adi Shamir** and **Leonard Adleman** in 1977.



Plaintext: GETREADY

Case 3: -

Plaintext – HELLOWORLD

Key – GEEKSFORGEEKS

Output –

Ciphertext: NIPVGBCIRH

2.

Plaintext: HELLOWORLD

Case 4: -

Plaintext – GOODBYE

Key – HELLO

Output –

Ciphertext: NSZOPFI

Plaintext: GOODBYE

```
import random

# Function to check if a number is prime

def is_prime(n):
    if n <= 1:
        return False  for i in
range(2, int(n**0.5) + 1):      if n
% i == 0:
        return False  return True

# Function to generate a prime number of specified
length

def
generate_prime(length):
    while True:
        prime_candidate =
random.randint(2**(length-1), 2**length - 1)
        if is_prime(prime_candidate):
            return prime_candidate
```

3.

Function to calculate the greatest common divisor (GCD) of two numbers

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

Function to find the modular inverse of a number

```
def mod_inverse(a, m):
   if gcd(a, m) != 1:
      return None  u1,
   u2, u3 = 1, 0, a
   v1, v2, v3 = 0, 1, m
      while v3 != 0:
           q = u3 // v3
v1, v2, v3, u1, u2, u3 = (
             u1 - q * v1,
      u2 - q * v2,          u3
- q * v3,            v1,
      v2,            v3,
                 )
      return u1 % m
```

Function to generate RSA keys

```
def generate_rsa_keys(key_length):
```

4.

```
# Generate two distinct prime numbers
p = generate_prime(key_length // 2)
q = generate_prime(key_length // 2)

# Compute modulus
modulus = p * q

# Compute Euler's totient function
phi = (p - 1) * (q - 1)

# Choose encryption exponent e (usually a small
prime number)
e = 65537

# Compute decryption exponent d
d = mod_inverse(e, phi)

return (e, modulus), (d, modulus)

# Function to encrypt a message using RSA


def encrypt(message, public_key):
e, modulus = public_key
encrypted = [pow(ord(c), e, modulus) for c in
message]
return encrypted

# Function to decrypt a message using RSA
```

5.

```
def decrypt(ciphertext, private_key):  d,
modulus = private_key  decrypted =
[chr(pow(c, d, modulus)) for c in ciphertext]
return ''.join(decrypted)


# Example usage message
= "HELLO"

# Generate RSA keys with a key length of 512 bits
public_key, private_key = generate_rsa_keys(512)

# Encrypt the message using the public key
encrypted_message = encrypt(message, public_key)

# Decrypt the ciphertext using the private key
decrypted_message = decrypt(encrypted_message,
private_key)

print("Original Message:", message)
print("Encrypted Message:", encrypted_message)
print("Decrypted Message:", decrypted_message)
```

2. Digital Signature Algorithm (DSA) :

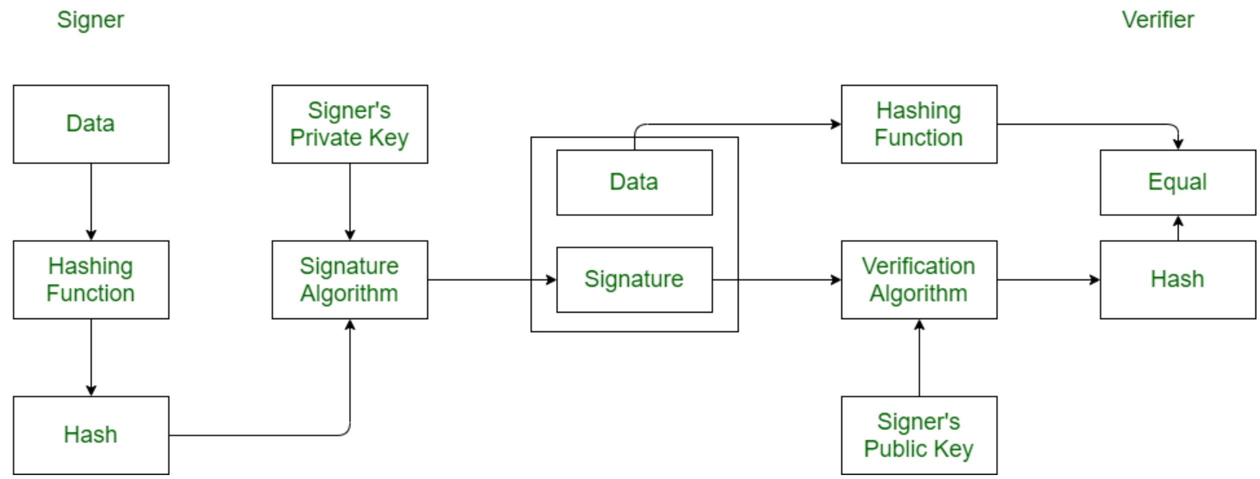
DSA stand for **Digital Signature Algorithm**. It is used for digital signature and its verification. It is based on mathematical concept of modular exponentiation and discrete logarithm. It was developed by **National Institute of Standards and Technology (NIST)** in 1991.

It involves four operations:

1. Key Generation
2. Key Distribution

6.

3. Signing
4. Signature Verification



Here's an example of how to implement the DSA (Digital Signature Algorithm) in Python using the '**cryptography**' library:

```
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.asymmetric
import dsa
from cryptography.hazmat.backends import
default_backend

# Key Generation
private_key = dsa.generate_private_key(
key_size=1024,
backend=default_backend()
)
public_key = private_key.public_key()

# Message message
= b"Hello, world!"
```

7.

```
# Signature Generation hash_algorithm
= hashes.SHA256()
signature = private_key.sign(
    message,
algorithm=hash_algorithm
    )

# Signature Verification try:
    public_key.verify(
    signature,
        message,
    algorithm=hash_algorithm
        )
    print("Signature is valid.")
except:  print("Signature is
invalid.")
```