Rashtriya Raksha University

School of Information Technology, Artificial Intelligence & Cyber Security (SITAICS)

At- Lavad, Dahegam, Gandhinagar, Gujarat-382305

Practical File

(Introduction to Cryptography)

Name: Sarthak Sanay
Enrollment No: 230031101611051
Subject Name: Introduction to Cryptography
Subject Code: G4A19ITC
Program: B.Tech CSE (with specialization in Cyber Security)
Year: 2nd year (Semester-IV)

This is certifying that Mr. Sarthak Sanay has satisfactorily completed all experiments in the practical work prescribed by SITAICS in the ITC laboratory.

Dr. Ashish Revar
SUBJECT INCHARGE

PRACTICAL - 8

AIM: TO IMPLEMENT THE DIFFIE HELLMAN KEY EXCHANGE ALGORITHM

BRIEF :-

The Diffie Hellman Key Exchange is a method that allows two people to securely share a secret key over a public channel. This means they can agree on a key for encryption without actually sending the key itself. It works by using mathematical operations based on large prime numbers and modular arithmetic. Each person picks a private number and then creates a public value using a shared base and prime. These public values are exchanged, and each person uses the other's public value along with their own private number to compute the same secret key.

This shared secret key can then be used for encrypted communication between the two parties. The strength of the Diffie-Hellman method lies in the difficulty of reversing the mathematical operations (a problem known as the Discrete Logarithm Problem). Even if someone sees the public values being shared, they can't easily figure out the secret key. However, Diffie-Hellman alone does not provide authentication, so it is often combined with other security methods to prevent attacks like man-in-the-middle.

ALGORITHM / PSEUDOCODE :-

```
print "Diffie-Hellman Key Exchange"
read sender_name
read receiver_name

read p
if p < 2 then
   error
end if

read g
if g ≤ 1 or g ≥ p or not is_primitive_root(g,p) then
    error
end if

read sender_priv
read receiver_priv
if sender_priv ≤ 0 or sender_priv ≥ p or receiver_priv ≤ 0 or receiver_priv ≥
p then
    error
end if

sender_pub -= mod_exp(g, sender_priv, p)
receiver_pub = mod_exp(g, receiver_priv, p)
sender_shared = mod_exp(receiver_pub, sender_priv, p)
receiver_shared = mod_exp(sender_pub, receiver_priv, p)

print sender_name + " pub:", sender_pub
print receiver_name + " pub:", receiver_pub
print sender_name + " shared:", sender_shared
print receiver_name + " shared:", receiver_shared

if sender_shared == receiver_shared then
    print "Shared key established"
else
    print "Error: keys mismatch"
end if

function mod_exp(b, e, m)
    return b^e mod m
end function

function is_primitive_root(g, p)
    return {g^k mod p | k←1..p-1} == {1..p-1}
end function
```

CODE :-

```
def mod_exp(base, exponent, modulus):
    return pow(base, exponent, modulus)

def is_primitive_root(g, p):
    required_set = set(num for num in range(1, p))
    actual_set = set(pow(g, powers, p) for powers in range(1, p))
    return required_set == actual_set

print("Diffie-Hellman Key Exchange :-\n")
sender_name = str(input("Enter sender's name: "))
receiver_name = str(input("Enter receiver's name: "))

p = int(input("\nEnter a large prime number (p): "))
if p < 2:
    raise ValueError("Prime number must be greater than 1.")

g = int(input(f"Enter a base number (generator) less than {p}: "))
if g <= 1 or g >= p:
    raise ValueError(f"Base number must be > 1 and < {p}.")
if not is_primitive_root(g, p):
    raise ValueError(f"{g} is not a valid generator (primitive root)
for {p}.")

sender_private = int(input(f"Enter {sender_name}'s private key (number
< p): "))
receiver_private = int(input(f"Enter {receiver_name}'s private key
(number < p): "))
if not (0 < sender_private < p and 0 < receiver_private < p):
    raise ValueError(f"Private keys must be between 1 and {p - 1}.")

sender_public = mod_exp(g, sender_private, p)
receiver_public = mod_exp(g, receiver_private, p)
sender_shared_key = mod_exp(receiver_public, sender_private, p)
receiver_shared_key = mod_exp(sender_public, receiver_private, p)

print("\nResults :-")
print(sender_name + "'s Public Key:", sender_public)
print(receiver_name + "'s Public Key:", receiver_public)
print(sender_name + "'s Shared Key:", sender_shared_key)
print(receiver_name + "'s Shared Key:", receiver_shared_key)

if sender_shared_key == receiver_shared_key:
    print("\nShared key successfully established!")
else:
    print("\nError: Shared keys do not match.")
```

OUTPUT :-

```
@sanaysarthak ➜/workspaces/crypto-lab/practicals (main) $ python diffie-hellman.py
Diffie-Hellman Key Exchange :-

Enter sender's name: Sarthak
Enter receiver's name: Sanay

Enter a large prime number (p): 23
Enter a base number (generator) less than 23: 11
Enter Sarthak's private key (number < p): 1
Enter Sanay's private key (number < p): 15

Results :-
Sarthak's Public Key: 11
Sanay's Public Key: 10
Sarthak's Shared Key: 10
Sanay's Shared Key: 10

Shared key successfully established!
@sanaysarthak ➜/workspaces/crypto-lab/practicals (main) $ python diffie-hellman.py
Diffie-Hellman Key Exchange :-

Enter sender's name: Elon
Enter receiver's name: Bezos

Enter a large prime number (p): 23
Enter a base number (generator) less than 23: 7
Enter Elon's private key (number < p): 11
Enter Bezos's private key (number < p): 14

Results :-
Elon's Public Key: 22
Bezos's Public Key: 2
Elon's Shared Key: 1
Bezos's Shared Key: 1

Shared key successfully established!
```