CRYPTOGRAPHY AND NETWORK SECURITY
LAB SESSION 10

ARYAN TRIVEDI
22BCE1979

QUESTION 1) Implement the following digital signature algorithms and verify the authentication and integrity of a text file.
ElGamal

SERVER SIDE CODE:

```
import socket
import hashlib
import random
from sympy import mod_inverse

def generate_keys():
    p = 593  # Large prime
    g = 123  # Primitive root
    x = random.randint(1, p - 2)
    y = pow(g, x, p)
    return p, g, x, y
```

```
def sign_message(p, g, x, message):
    h = int(hashlib.sha256(message.encode()).hexdigest(), 16)
    while True:
        k = random.randint(1, p - 2)
        if gcd(k, p - 1) == 1:
            break
    r = pow(g, k, p)
    k_inv = mod_inverse(k, p - 1)
    s = (k_inv * (h - x * r)) % (p - 1)
    return r, s
```

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

```python
def start_server():
    host = 'localhost'
    port = 5000
    p, g, x, y = generate_keys()

    # Read message from file
    with open("a.txt", "r") as file:
        message = file.read().strip()

    r, s = sign_message(p, g, x, message)

    with socket.socket() as server:
        server.bind((host, port))
        server.listen(1)
        print("Server listening...")

        conn, addr = server.accept()
        with conn:
            print(f"Connected by {addr}")
            data = {
                "message": message,
                "signature": (r, s),
                "public_key": (p, g, y)
            }
            conn.send(str(data).encode())

if __name__ == '__main__':
    start_server()
```

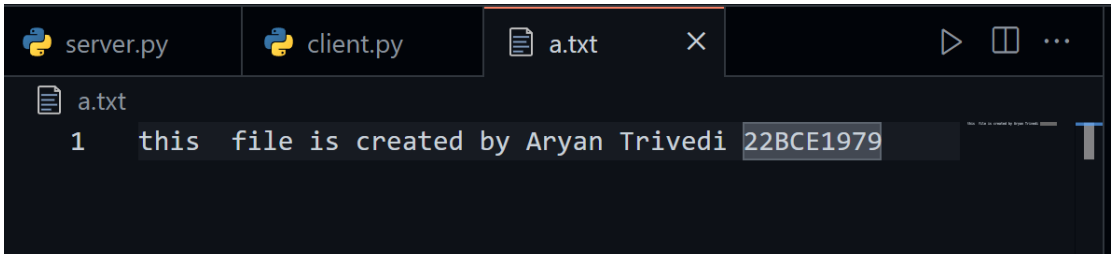
CLIENT SIDE CODE:

```python
import socket
import hashlib

def verify_signature(message, r, s, p, g, y):
    h = int(hashlib.sha256(message.encode()).hexdigest(), 16)
    v1 = (pow(y, r, p) * pow(r, s, p)) % p
    v2 = pow(g, h, p)
    return v1 == v2

def start_client():
    host = 'localhost'
    port = 5000

    with socket.socket() as client:
        client.connect((host, port))
        data = client.recv(4096).decode()
        data = eval(data)

        message = data["message"]
        r, s = data["signature"]
        p, g, y = data["public_key"]

        print("Received message:", message)
        print("Signature (r, s):", r, s)
        print("Public key (p, g, y):", p, g, y)

        valid = verify_signature(message, r, s, p, g, y)
        if valid:
            print("✅Signature is valid. Integrity and authenticity confirmed.")
        else:
            print("❌Signature is INVALID. Message may have been tampered.")
```

```python
if __name__ == '__main__':
    start_client()
```

OUTPUTS:

IMPLEMENTED USING A TEXT FILE NAMED a.txt



SERVER SIDE OUTPUT:

```
PS C:\Users\Aryan\Desktop\crypto lab>python server.py
Server listening...
Connected by ('127.0.0.1', 61288)
PS C:\Users\Aryan\Desktop\crypto lab>
```

CLIENT SIDE OUTPUT:

```
PS C:\Users\Aryan\Desktop\crypto lab> python client.py
Received message: this  file is created by Aryan Trivedi 22BCE1979
Signature (r, s): 510 6
Public key (p, g, y): 593 123 168
✅ Signature is valid. Integrity and authenticity confirmed.
PS C:\Users\Aryan\Desktop\crypto lab>
```

FINAL RESULT: SIGNATURE IS VALID. INTEGRITY AND AUTHENTICITY ARE MAINTAINED

QUESTION 2) Implement the following digital signature algorithms and verify the authentication and integrity of a text file.
DSS

SERVER SIDE CODE:

```
import socket
import hashlib
import random

# Global Parameters for DSA
p = 1019  # Large prime
q = 157   # Prime divisor of (p-1)
g = 3     # Generator of subgroup of order q
```

```
# Read message from file
def read_message():
    with open("a.txt", "r") as file:
        return file.read().strip()
```

```
# Generate DSA keys
def generate_keys():
    x = random.randint(1, q - 1)  # Private key
    y = pow(g, x, p)  # Public key
    return x, y
```

```
# DSA signing
def sign_message(message, x):
    hash_value = int(hashlib.sha256(message.encode()).hexdigest(), 16) % q
```

```
    while True:
        k = random.randint(1, q - 1)
        if gcd(k, q) == 1:
            break
```

```
    r = pow(g, k, p) % q
    k_inv = pow(k, -1, q)  # Modular inverse
    s = (k_inv * (hash_value + x * r)) % q
```

```
    return r, s, hash_value
```

```
# GCD function
def gcd(a, b):
    while b:
        a, b = b, a % b
    return a
```

```
def start_server():
    host = 'localhost'
    port = 5002
    message = read_message()
```

```
    print("\n[Server] Original Message from a.txt:")
    print(message)
```

```
    x, y = generate_keys()
    r, s, hash_value = sign_message(message, x)
```

```
    print("\n[Server] SHA256 Hash of message (mod q):", hash_value)
    print("\n[Server] Signature (r, s):", r, s)
    print("\n[Server] Public Key (p, q, g, y):", p, q, g, y)
```

```
    with socket.socket() as server:
        server.bind((host, port))
        server.listen(1)
        print("\n[Server] Listening on port", port)
```

```
        conn, addr = server.accept()
        with conn:
            print(f"\n[Server] Connected by {addr}")
            data = {
                "message": message,
                "signature": (r, s),
                "public_key": (p, q, g, y)
            }
            conn.send(str(data).encode())
```

```
if __name__ == "__main__":
    start_server()
```

CLIENT SIDE CODE:

```
import socket
import hashlib
```

```
# DSA signature verification
def verify_signature(message, r, s, p, q, g, y):
    hash_value = int(hashlib.sha256(message.encode()).hexdigest(), 16) % q
```

```
    w = pow(s, -1, q)
    u1 = (hash_value * w) % q
    u2 = (r * w) % q
    v = ((pow(g, u1, p) * pow(y, u2, p)) % p) % q
```

```
    print("\n[Client] SHA256 Hash of received message (mod q):", hash_value)
    print("\n[Client] Calculated v:", v)
```

```
    return v == r
```

```
def start_client():
    host = 'localhost'
    port = 5002
```

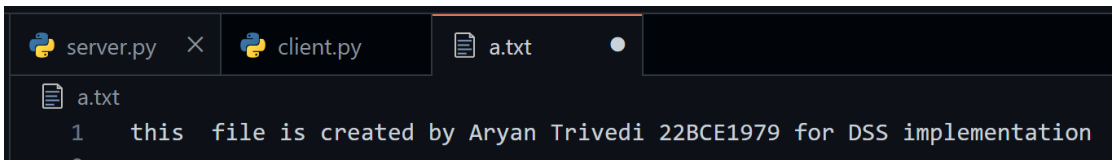
```
    with socket.socket() as client:
        client.connect((host, port))
        received = client.recv(8192).decode()
        data = eval(received)
```

```
        message = data["message"]
        r, s = data["signature"]
```

```
        p, q, g, y = data["public_key"]

        print("\n[Client] Received Message:")
        print(message)

        print("\n[Client] Received Signature (r, s):", r, s)
        print("\n[Client] Received Public Key (p, q, g, y):", p, q, g, y)

        if verify_signature(message, r, s, p, q, g, y):
            print("\n✅Signature is VALID. Message is authentic and untampered.")
        else:
            print("\n❌Signature is INVALID. Message integrity or authenticity failed.")

if __name__ == "__main__":
    start_client()
```

OUTPUTS:

IMPLEMENTED USING TEXT FILE NAMED a.txt



SERVER SIDE OUTPUT:

```
[Server] Original Message from a.txt:
this  file is created by Aryan Trivedi 22BCE1979 for

[Server] Original Message from a.txt:


 22BCE1979 for DSS implementation
 22BCE1979 for DSS implementation

[Server] SHA256 Hash of message (mod q) 22BCE1979 for DSS implementation

[Server] SHA256 Hash of message (mod q) 22BCE1979 for DSS implementation

[Server] SHA256 Hash of message (mod q) 22BCE1979 for DSS implementation

[Server] SHA256 Hash of message (mod q) 22BCE1979 for DSS implementation

 22BCE1979 for DSS implementation        91
```

CLIENT SIDE OUTPUT:

FINAL RESULT: SIGNATURE IS VALID. INTEGRITY AND AUTHENTICITY ARE MAINTAINED