

# DSA Algorithm

## Cryptography – Lab 8

Name: Ojas Patil  
Reg No: 21BAI1106

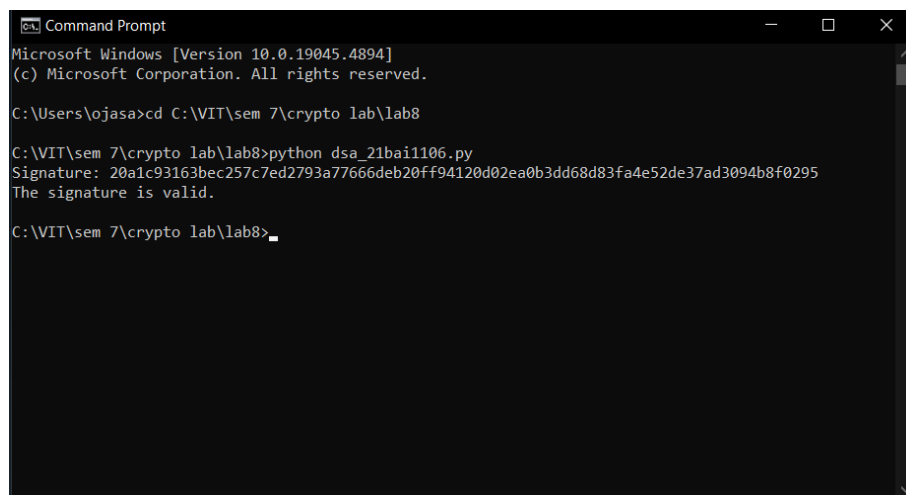
### Task

To Develop a Python-based DSA (Digital Signature Algorithm) implementation.

### SHA512 Algorithm - Definition

The **Digital Signature Algorithm (DSA)** is an asymmetric cryptographic method used to create digital signatures, ensuring message authenticity and integrity. It involves key pair generation, message signing with a private key, and signature verification using the public key. DSA operates through modular arithmetic and discrete logarithms, offering security without encryption or key exchange functions.

### SHA512 Algorithm - Output Snapshot



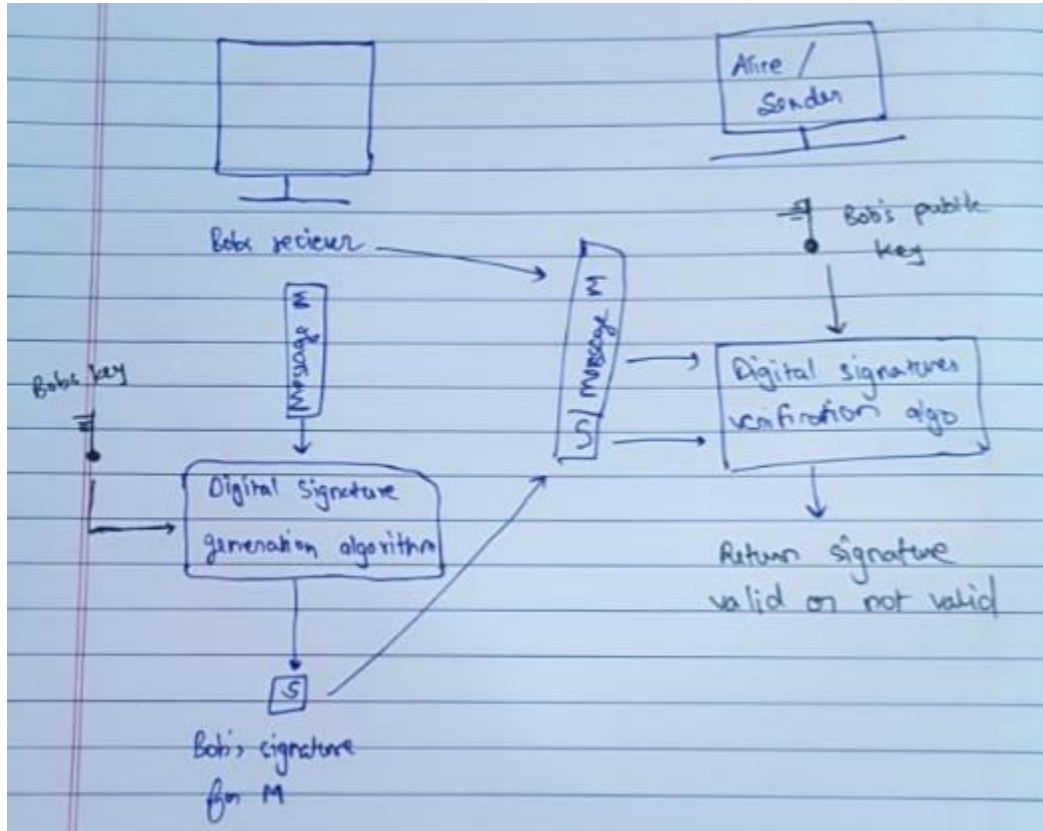
```
Command Prompt
Microsoft Windows [Version 10.0.19045.4894]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ojasa>cd C:\VIT\sem 7\crypto lab\lab8

C:\VIT\sem 7\crypto lab\lab8>python dsa_21bai1106.py
Signature: 20a1c93163bec257c7ed2793a77666deb20ff94120d02ea0b3dd68d83fa4e52de37ad3094b8f0295
The signature is valid.

C:\VIT\sem 7\crypto lab\lab8>
```

## SHA512 - Diagram



## Source Code

```
from Crypto.PublicKey import DSA
from Crypto.Signature import DSS
from Crypto.Hash import SHA256

def generate_keys():
    private_key = DSA.generate(1024)
    public_key = private_key.publickey()
    return private_key, public_key

def sign_message(private_key, message):
    hash_obj = SHA256.new(message.encode('utf-8'))
    signer = DSS.new(private_key, 'fips-186-3')
    signature = signer.sign(hash_obj)
    return signature

# Verify the signature using the public key
def verify_signature(public_key, message, signature):
```

```
hash_obj = SHA256.new(message.encode('utf-8'))
verifier = DSS.new(public_key, 'fips-186-3')
try:
    verifier.verify(hash_obj, signature)
    return True
except ValueError:
    return False

if __name__ == "__main__":
    message = "This is a test message for DSA."

    private_key, public_key = generate_keys()

    # Sign the message using the private key
    signature = sign_message(private_key, message)
    print("Signature:", signature.hex())

    # Verify the signature using the public key
    is_valid = verify_signature(public_key, message, signature)
    if is_valid:
        print("The signature is valid.")
    else:
        print("The signature is invalid.")
```

## **Conclusion**

The Python-based DSA implementation demonstrates key generation, signing, and verification, highlighting DSA's role in ensuring message authenticity and integrity. This algorithm is crucial for cryptographic applications where secure digital signatures are needed.