CRYPTOGRAPHY AND NETWORK SECURITY

LAB 12 : DIGITAL SIGNATURES [DSS AND ELGAMMAL]

NAME : OM SUBRATO DEY

REGISTER NO. : 21BAI1876

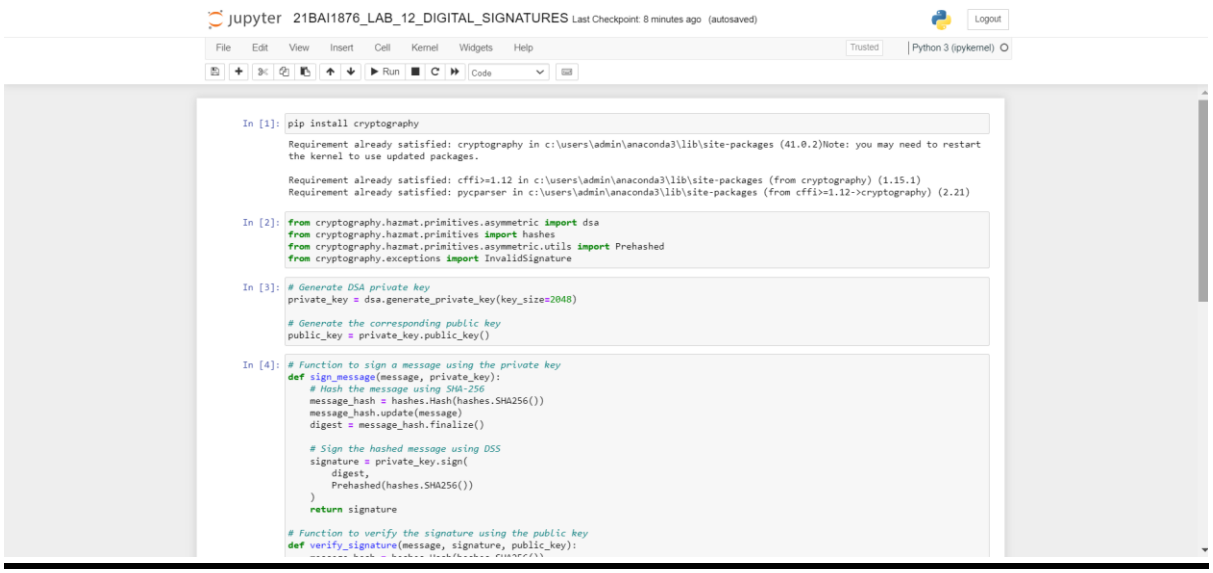
1. DSS DIGITAL SIGNATURE

CODE:

```
from cryptography.hazmat.primitives.asymmetric import dsa
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.asymmetric.utils import Prehashed
from cryptography.exceptions import InvalidSignature

# Generate DSA private key
private_key = dsa.generate_private_key(key_size=2048)

# Generate the corresponding public key
public_key = private_key.public_key()

# Function to sign a message using the private key
def sign_message(message, private_key):
    # Hash the message using SHA-256
    message_hash = hashes.Hash(hashes.SHA256())
    message_hash.update(message)
    digest = message_hash.finalize()

    # Sign the hashed message using DSS
    signature = private_key.sign(
        digest,
        Prehashed(hashes.SHA256())
    )
    return signature
```

```
# Function to verify the signature using the public key
def verify_signature(message, signature, public_key):
    message_hash = hashes.Hash(hashes.SHA256())
    message_hash.update(message)
    digest = message_hash.finalize()

    try:
        # Verify the signature
        public_key.verify(
            signature,
            digest,
            Prehashed(hashes.SHA256())
        )
        return True
    except InvalidSignature:
        return False

# Example usage
message = b"This is a secure message."
signature = sign_message(message, private_key)

# Verification
is_valid = verify_signature(message, signature, public_key)

if is_valid:
    print("Signature is valid!")
else:
    print("Signature is invalid.")
```

OUTPUT:


jupyter 21BAI1876_LAB_12_DIGITAL_SIGNATURES Last Checkpoint: 8 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3 (ipykernel)

Run Code

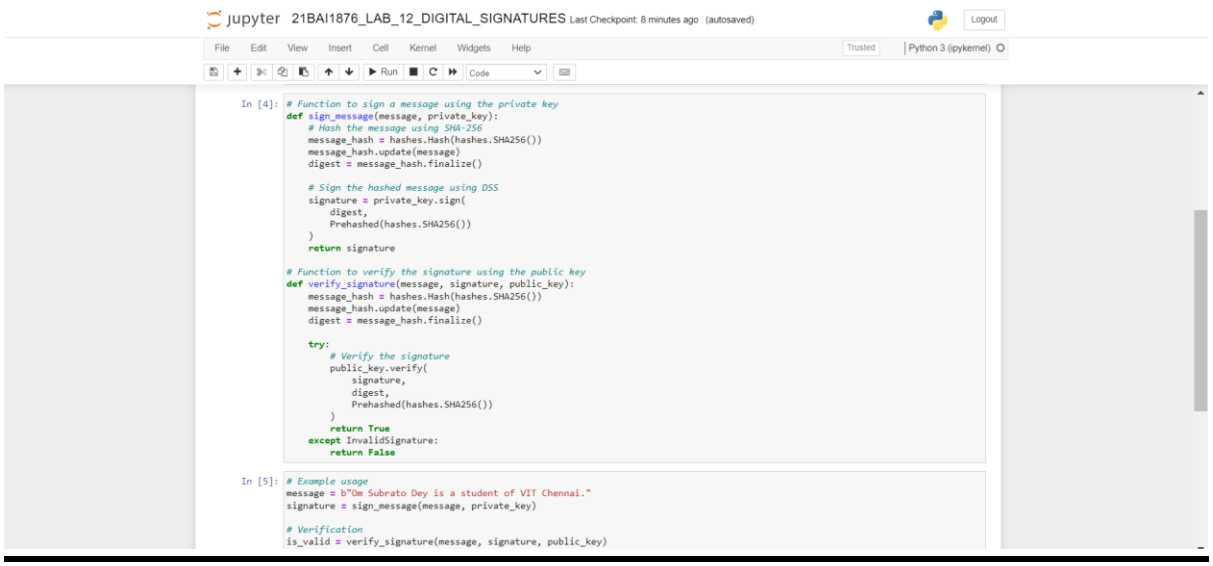
In [1]:
```
pip install cryptography
```

```
Requirement already satisfied: cryptography in c:\users\admin\anaconda3\lib\site-packages (41.0.2)Note: you may need to restart
the kernel to use updated packages.

Requirement already satisfied: cffi>=1.12 in c:\users\admin\anaconda3\lib\site-packages (from cryptography) (1.15.1)
Requirement already satisfied: pycparser in c:\users\admin\anaconda3\lib\site-packages (from cffi>=1.12->cryptography) (2.21)
```

In [2]:
```
from cryptography.hazmat.primitives.asymmetric import dsa
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.asymmetric.utils import Prehashed
from cryptography.exceptions import InvalidSignature
```

In [3]:
```
# Generate DSA private key
private_key = dsa.generate_private_key(key_size=2048)

# Generate the corresponding public key
public_key = private_key.public_key()
```

In [4]:
```
# Function to sign a message using the private key
def sign_message(message, private_key):
    # Hash the message using SHA-256
    message_hash = hashes.Hash(hashes.SHA256())
    message_hash.update(message)
    digest = message_hash.finalize()

    # Sign the hashed message using DSS
    signature = private_key.sign(
        digest,
        Prehashed(hashes.SHA256())
    )
    return signature

# Function to verify the signature using the public key
def verify_signature(message, signature, public_key):
```


jupyter 21BAI1876_LAB_12_DIGITAL_SIGNATURES Last Checkpoint: 8 minutes ago (autosaved)

Logout

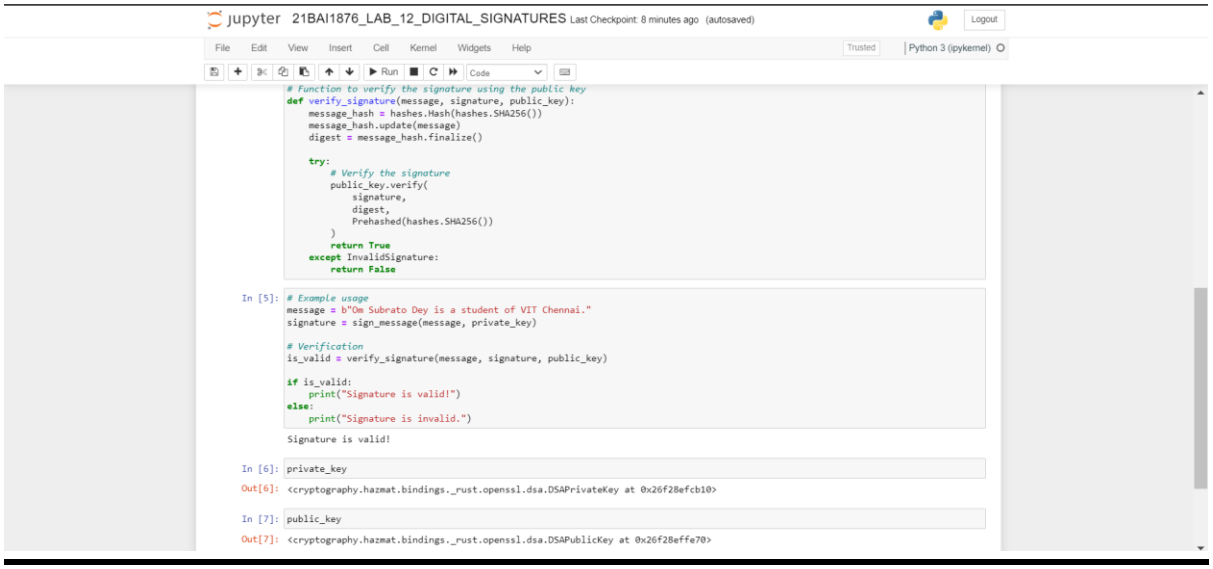
File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3 (ipykernel)

Run Code

In [4]:
```
# Function to sign a message using the private key
def sign_message(message, private_key):
    # Hash the message using SHA-256
    message_hash = hashes.Hash(hashes.SHA256())
    message_hash.update(message)
    digest = message_hash.finalize()

    # Sign the hashed message using DSS
    signature = private_key.sign(
        digest,
        Prehashed(hashes.SHA256())
    )
    return signature

# Function to verify the signature using the public key
def verify_signature(message, signature, public_key):
    message_hash = hashes.Hash(hashes.SHA256())
    message_hash.update(message)
    digest = message_hash.finalize()

    try:
        # Verify the signature
        public_key.verify(
            signature,
            digest,
            Prehashed(hashes.SHA256())
        )
        return True
    except InvalidSignature:
        return False
```

In [5]:
```
# Example usage
message = b"Om Subrato Dey is a student of VIT Chennai."
signature = sign_message(message, private_key)

# Verification
is_valid = verify_signature(message, signature, public_key)
```


jupyter 21BAI1876_LAB_12_DIGITAL_SIGNATURES Last Checkpoint: 8 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3 (ipykernel)

Run Code

```
# Function to verify the signature using the public key
def verify_signature(message, signature, public_key):
    message_hash = hashes.Hash(hashes.SHA256())
    message_hash.update(message)
    digest = message_hash.finalize()

    try:
        # Verify the signature
        public_key.verify(
            signature,
            digest,
            Prehashed(hashes.SHA256())
        )
        return True
    except InvalidSignature:
        return False
```

In [5]:
```
# Example usage
message = b"Om Subrato Dey is a student of VIT Chennai."
signature = sign_message(message, private_key)

# Verification
is_valid = verify_signature(message, signature, public_key)

if is_valid:
    print("Signature is valid!")
else:
    print("Signature is invalid.")
```

```
Signature is valid!
```

In [6]:
```
private_key
```

Out[6]:
```
<cryptography.hazmat.bindings._rust.openssl.dsa.DSAPrivateKey at 0x26f28efcb10>
```

In [7]:
```
public_key
```

Out[7]:
```
<cryptography.hazmat.bindings._rust.openssl.dsa.DSAPublicKey at 0x26f28effe70>
```

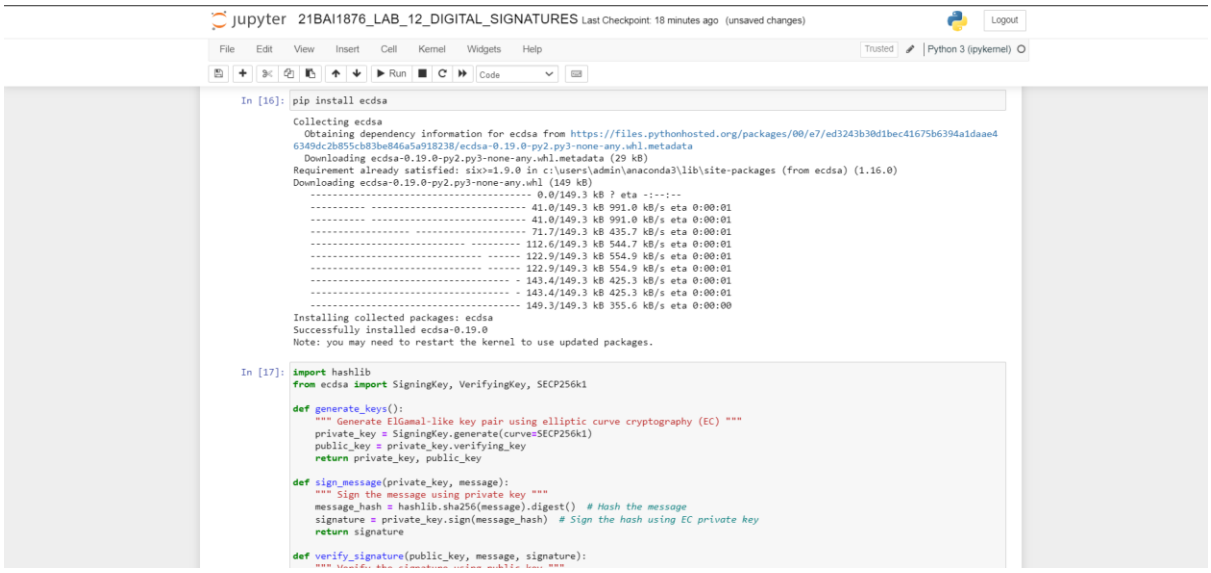
2. ELGAMMAL DIGITAL SIGNATURE:

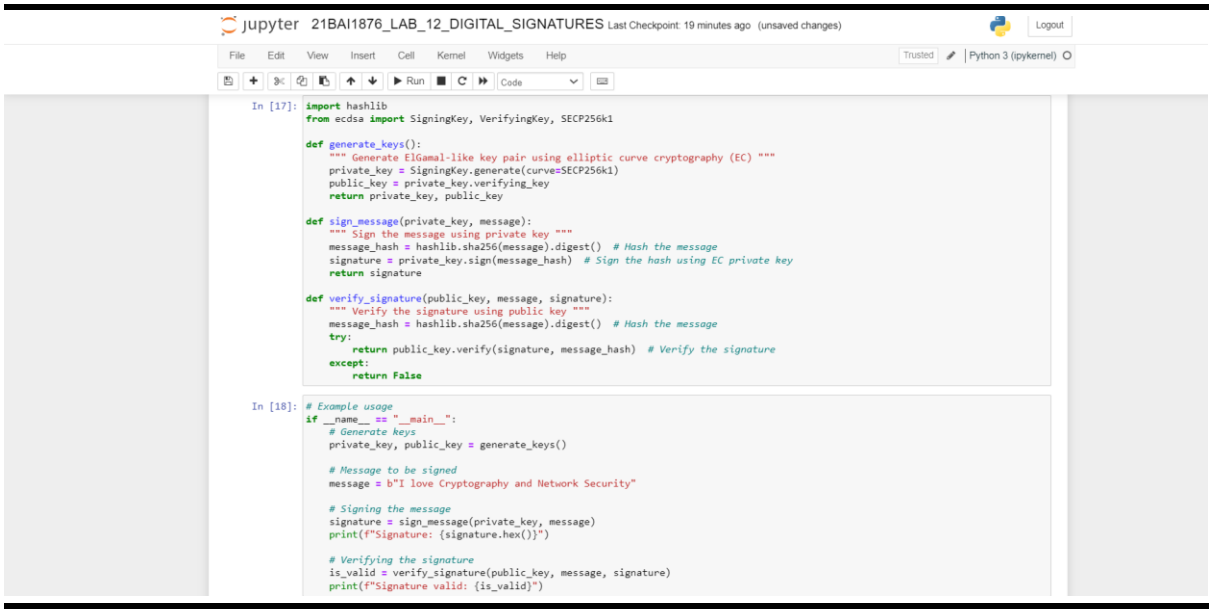
CODE:

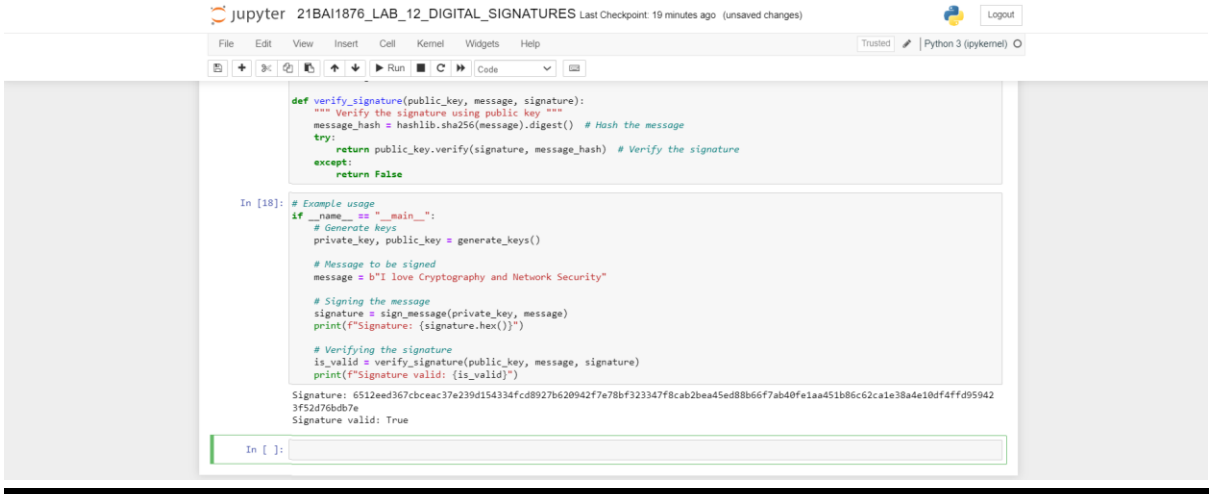
```
import hashlib
from ecdsa import SigningKey, VerifyingKey, SECP256k1


def generate_keys():
    """ Generate ElGamal-like key pair using elliptic curve cryptography (EC) """
    private_key = SigningKey.generate(curve=SECP256k1)
    public_key = private_key.verifying_key
    return private_key, public_key


def sign_message(private_key, message):
    """ Sign the message using private key """
    message_hash = hashlib.sha256(message).digest()  # Hash the message
    signature = private_key.sign(message_hash)  # Sign the hash using EC private key
    return signature


def verify_signature(public_key, message, signature):
    """ Verify the signature using public key """
    message_hash = hashlib.sha256(message).digest()  # Hash the message
    try:
        return public_key.verify(signature, message_hash)  # Verify the signature
    except:
        return False
```

```
# Example usage
if __name__ == "__main__":
    # Generate keys
    private_key, public_key = generate_keys()

    # Message to be signed
    message = b"Elliptic Curve ElGamal Digital Signature
Example"

    # Signing the message
    signature = sign_message(private_key, message)
    print(f"Signature: {signature.hex()}")

    # Verifying the signature
    is_valid = verify_signature(public_key, message,
signature)
    print(f"Signature valid: {is_valid}")
```

OUTPUT:


jupyter 21BAI1876_LAB_12_DIGITAL_SIGNATURES Last Checkpoint: 18 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted Python 3 (ipykernel)
Run Code
In [16]: pip install ecdsa
Collecting ecdsa
Obtaining dependency information for ecdsa from https://files.pythonhosted.org/packages/00/e7/ed3243b30d1bec41675b6394a1daae46349dc2b855cb83be846a5a91823 8/ecdsa-0.19.0-py2.py3-none-any.whl.metadata
Downloading ecdsa-0.19.0-py2.py3-none-any.whl.metadata (29 kB)
Requirement already satisfied: six>=1.9.0 in c:\users\admin\anaconda3\lib\site-packages (from ecdsa) (1.16.0)
Downloading ecdsa-0.19.0-py2.py3-none-any.whl (149 kB)
0.0/149.3 kB ? eta -:--:--
41.0/149.3 kB 991.0 kB/s eta 0:00:01
41.0/149.3 kB 991.0 kB/s eta 0:00:01
71.7/149.3 kB 435.7 kB/s eta 0:00:01
112.6/149.3 kB 544.7 kB/s eta 0:00:01
122.9/149.3 kB 554.9 kB/s eta 0:00:01
122.9/149.3 kB 554.9 kB/s eta 0:00:01
143.4/149.3 kB 425.3 kB/s eta 0:00:01
143.4/149.3 kB 425.3 kB/s eta 0:00:01
149.3/149.3 kB 355.6 kB/s eta 0:00:00
Installing collected packages: ecdsa
Successfully installed ecdsa-0.19.0
Note: you may need to restart the kernel to use updated packages.
In [17]: import hashlib
from ecdsa import SigningKey, VerifyingKey, SECP256k1
def generate_keys():
""" Generate ElGamal-like key pair using elliptic curve cryptography (EC) """
private_key = SigningKey.generate(curve=SECP256k1)
public_key = private_key.verifying_key
return private_key, public_key
def sign_message(private_key, message):
""" Sign the message using private key """
message_hash = hashlib.sha256(message).digest() # Hash the message
signature = private_key.sign(message_hash) # Sign the hash using EC private key
return signature
def verify_signature(public_key, message, signature):


jupyter 21BAI1876_LAB_12_DIGITAL_SIGNATURES Last Checkpoint: 19 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted Python 3 (ipykernel)
Run Code
In [17]: import hashlib
from ecdsa import SigningKey, VerifyingKey, SECP256k1
def generate_keys():
""" Generate ElGamal-like key pair using elliptic curve cryptography (EC) """
private_key = SigningKey.generate(curve=SECP256k1)
public_key = private_key.verifying_key
return private_key, public_key
def sign_message(private_key, message):
""" Sign the message using private key """
message_hash = hashlib.sha256(message).digest() # Hash the message
signature = private_key.sign(message_hash) # Sign the hash using EC private key
return signature
def verify_signature(public_key, message, signature):
""" Verify the signature using public key """
message_hash = hashlib.sha256(message).digest() # Hash the message
try:
return public_key.verify(signature, message_hash) # Verify the signature
except:
return False
In [18]: # Example usage
if __name__ == "__main__":
# Generate keys
private_key, public_key = generate_keys()
# Message to be signed
message = b"I love Cryptography and Network Security"
# Signing the message
signature = sign_message(private_key, message)
print(f"Signature: {signature.hex()}")
# Verifying the signature
is_valid = verify_signature(public_key, message, signature)
print(f"Signature valid: {is_valid}")


jupyter 21BAI1876_LAB_12_DIGITAL_SIGNATURES Last Checkpoint: 19 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted Python 3 (ipykernel)
Run Code
def verify_signature(public_key, message, signature):
""" Verify the signature using public key """
message_hash = hashlib.sha256(message).digest() # Hash the message
try:
return public_key.verify(signature, message_hash) # Verify the signature
except:
return False
In [18]: # Example usage
if __name__ == "__main__":
# Generate keys
private_key, public_key = generate_keys()
# Message to be signed
message = b"I love Cryptography and Network Security"
# Signing the message
signature = sign_message(private_key, message)
print(f"Signature: {signature.hex()}")
# Verifying the signature
is_valid = verify_signature(public_key, message, signature)
print(f"Signature valid: {is_valid}")
Signature: 6512eed367cbceac37e239d154334fcd8927b620942f7e78bf323347f8cab2bea45ed88b66f7ab40fe1aa451b86c62ca1e38a4e10df4ffd959423f52d76bdb7e
Signature valid: True
In [ ]: