**NETWORK SECURITY ASSIGNMENT**

AES (Advanced Encryption Standard) Algorithm

Name: K Surya Sunadh

Roll no: CS22B1020

**Introduction**

**AES (Advanced Encryption Standard)**: The modern symmetric-key standard, adopted in 2001. It uses 128, 192, or 256-bit keys and is widely used today (e.g., in HTTPS, VPNs, and disk encryption) because it’s secure and efficient.

I’ll implement AES since it’s widely used and a good balance between practicality and security. Below is a simple Python implementation of AES encryption and decryption using the pycryptodome library, which is a robust cryptographic toolkit. I’ll use AES in CBC (Cipher Block Chaining) mode with a 256-bit key, as it’s a common setup.

**Code:**

from Crypto.Cipher import AES

from Crypto.Random import get\_random\_bytes

from Crypto.Util.Padding import pad, unpad

def aes\_encrypt(plaintext, key):

# Generate a random 16-byte IV (Initialization Vector)

iv = get\_random\_bytes(16)

# Create AES cipher object in CBC mode

cipher = AES.new(key, AES.MODE\_CBC, iv)

# Pad the plaintext to be a multiple of 16 bytes

padded\_text = pad(plaintext.encode('utf-8'), AES.block\_size)

# Encrypt the padded plaintext

ciphertext = cipher.encrypt(padded\_text)

# Return IV concatenated with ciphertext (IV is needed for decryption)

return iv + ciphertext

def aes\_decrypt(ciphertext, key):

# Extract the IV from the first 16 bytes

iv = ciphertext[:16]

# The actual encrypted data is after the IV

encrypted\_data = ciphertext[16:]

# Create AES cipher object in CBC mode with the same IV

cipher = AES.new(key, AES.MODE\_CBC, iv)

# Decrypt and remove padding

padded\_plaintext = cipher.decrypt(encrypted\_data)

plaintext = unpad(padded\_plaintext, AES.block\_size)

return plaintext.decode('utf-8')

# Example usage

if \_\_name\_\_ == "\_\_main\_\_":

# Generate a random 32-byte (256-bit) key

key = get\_random\_bytes(32)

# Sample plaintext

message = "Hello, this is a secret message!"

# Encrypt

encrypted = aes\_encrypt(message, key)

print("Encrypted (hex):", encrypted.hex())

# Decrypt

decrypted = aes\_decrypt(encrypted, key)

print("Decrypted:", decrypted)

**Explanation:**

**Key**: A 256-bit (32-byte) key is randomly generated. In practice, you’d securely store or derive this key.

**IV**: A 16-byte random Initialization Vector ensures each encryption is unique, even with the same key and plaintext.

**Padding**: AES requires data in 16-byte blocks, so we pad the plaintext.

**CBC Mode**: Each block is XORed with the previous encrypted block (or IV for the first), adding security.

**Output**: The encrypted result includes the IV (needed for decryption) followed by the ciphertext.

**Output:**

