

**SECURE VPN COMMUNICATION USING AES**

**ENCRYPTION ALGORITHM**

**COMPUTER**

**NETWORK**

**SUBMITTED BY**

SITHARTH S

GURUSARAN S

BENNY CHRISTIYAN.J

III CSE B

**COMPUTER NEWORKS**

1. **Generate Secure VPN Communication using AES encrytion algorithm**

**using PYTHON**

**Client Program**:

import socket

import webbrowser

from cryptography.hazmat.primitives.asymmetric import ec

from cryptography.hazmat.primitives import serialization

from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes

from cryptography.hazmat.primitives import hashes

from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC

import os

import tkinter as tk

from tkinter import simpledialog

import time

import sys

# Generate a new ECDH key pair for the client

private\_key = ec.generate\_private\_key(ec.SECP256R1())

public\_key = private\_key.public\_key()

# Serialize public key to send to the server

def serialize\_public\_key(public\_key):

return public\_key.public\_bytes(

encoding=serialization.Encoding.PEM,

format=serialization.PublicFormat.SubjectPublicKeyInfo

)

# Deserialize the server's public key

def deserialize\_public\_key(pem\_data):

return serialization.load\_pem\_public\_key(pem\_data)

# Encrypt data using AES-GCM

def encrypt\_data(key, data):

nonce = os.urandom(12)

cipher = Cipher(algorithms.AES(key), modes.GCM(nonce))

encryptor = cipher.encryptor()

encrypted\_data = encryptor.update(data) + encryptor.finalize()

return nonce + encryptor.tag + encrypted\_data

# Decrypt data using AES-GCM

def decrypt\_data(key, encrypted\_data):

nonce = encrypted\_data[:12]

tag = encrypted\_data[12:28]

ciphertext = encrypted\_data[28:]

cipher = Cipher(algorithms.AES(key), modes.GCM(nonce, tag))

decryptor = cipher.decryptor()

return decryptor.update(ciphertext) + decryptor.finalize()

# Function to handle key entry and validation

def verification(expected\_key, max\_attempts=3, retry\_delay=10):

attempts = 0

while attempts < max\_attempts:

root = tk.Tk()

root.withdraw() # Hide the main tkinter window

# Mask the input text like a password

user\_input = simpledialog.askstring("Shared Key", "Enter the shared key:", show='\*')

if user\_input is None:

print("Operation canceled by the user.")

sys.exit()

if user\_input == expected\_key:

print("Key entered correctly!")

return True

else:

attempts += 1

print(f"Incorrect key. {max\_attempts - attempts} attempts left.")

root.destroy()

# Countdown after reaching maximum attempts

print(f"Maximum attempts reached. Please wait {retry\_delay} seconds before retrying.")

for i in range(retry\_delay, 0, -1):

sys.stdout.write(f"\rRetrying in {i} seconds...")

sys.stdout.flush()

time.sleep(1)

# Clear the countdown message from the terminal

sys.stdout.write("\r" + " " \* 50 + "\r")

sys.stdout.flush()

print("\n")

return False

# Client setup

def start\_client(server\_ip='127.0.0.1', port=65432, request\_url='http://google.com'):

client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

client\_socket.connect((server\_ip, port))

print("Connected with server")

# Send client's public key

client\_socket.send(serialize\_public\_key(public\_key))

# Receive server's public key

server\_public\_key\_pem = client\_socket.recv(1024)

server\_public\_key = deserialize\_public\_key(server\_public\_key\_pem)

# Generate shared key

shared\_secret = private\_key.exchange(ec.ECDH(), server\_public\_key)

key = PBKDF2HMAC(

algorithm=hashes.SHA256(),

length=32,

salt=b'salt',

iterations=100000,

).derive(shared\_secret)

shared\_key\_hex = key.hex()

print("Shared key:", shared\_key\_hex)

# Wait for the correct key entry

while not verification(shared\_key\_hex):

continue

print("Key exchange successful")

# Encrypt and send the request

encrypted\_request = encrypt\_data(key, request\_url.encode())

print("Encrypted request to server:", encrypted\_request)

client\_socket.send(encrypted\_request)

# Receive and decrypt the response

data = b""

while True:

part = client\_socket.recv(4096)

if not part: # No more data

break

data += part

encrypted\_response = data

print("Encrypted response from server:", encrypted\_response)

decrypted\_response = decrypt\_data(key, encrypted\_response).decode()

# Close the client socket

client\_socket.close()

# Open the decrypted URL in a web browser

if decrypted\_response.startswith('http'):

webbrowser.open(decrypted\_response)

else:

print("Received response is not a valid URL.")

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

start\_client(request\_url='http://google.com')

**Client Program**:

import socket

import requests

from cryptography.hazmat.primitives.asymmetric import ec

from cryptography.hazmat.primitives import serialization

from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes

from cryptography.hazmat.primitives import hashes

from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC

import os

# Generate a new ECDH key pair for the server

private\_key = ec.generate\_private\_key(ec.SECP256R1())

public\_key = private\_key.public\_key()

# Serialize public key to send to the client

def serialize\_public\_key(public\_key):

return public\_key.public\_bytes(

encoding=serialization.Encoding.PEM,

format=serialization.PublicFormat.SubjectPublicKeyInfo

)

# Deserialize the client's public key

def deserialize\_public\_key(pem\_data):

return serialization.load\_pem\_public\_key(pem\_data)

# Encrypt data using AES-GCM

def encrypt\_data(key, data):

nonce = os.urandom(12)

cipher = Cipher(algorithms.AES(key), modes.GCM(nonce))

encryptor = cipher.encryptor()

encrypted\_data = encryptor.update(data) + encryptor.finalize()

return nonce + encryptor.tag + encrypted\_data

# Decrypt data using AES-GCM

def decrypt\_data(key, encrypted\_data):

nonce = encrypted\_data[:12]

tag = encrypted\_data[12:28]

ciphertext = encrypted\_data[28:]

cipher = Cipher(algorithms.AES(key), modes.GCM(nonce, tag))

decryptor = cipher.decryptor()

return decryptor.update(ciphertext) + decryptor.finalize()

# Handle client connection

def handle\_client(client\_socket):

print(f"Connection from {client\_socket.getpeername()}")

# Send server's public key

client\_socket.send(serialize\_public\_key(public\_key))

# Receive client's public key

client\_public\_key\_pem = client\_socket.recv(1024)

client\_public\_key = deserialize\_public\_key(client\_public\_key\_pem)

# Generate shared key

shared\_secret = private\_key.exchange(ec.ECDH(), client\_public\_key)

key = PBKDF2HMAC(

algorithm=hashes.SHA256(),

length=32,

salt=b'salt',

iterations=100000,

).derive(shared\_secret)

#print("Key exchange successful")

# Receive encrypted request

encrypted\_request = client\_socket.recv(4096)

print(f"Encrypted request from client: {encrypted\_request}")

decrypted\_request = decrypt\_data(key, encrypted\_request).decode()

print(f"Decrypted request from client: {decrypted\_request}")

# Fetch the actual content

response = requests.get(decrypted\_request)

# Encrypt and send the response

encrypted\_response = encrypt\_data(key, response.url.encode())

client\_socket.send(encrypted\_response)

client\_socket.close()

# Server setup

def start\_server(host='0.0.0.0', port=8000):

server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

server\_socket.bind((host, port))

server\_socket.listen(5)

print(f"Server listening on {host}:{port}")

while True:

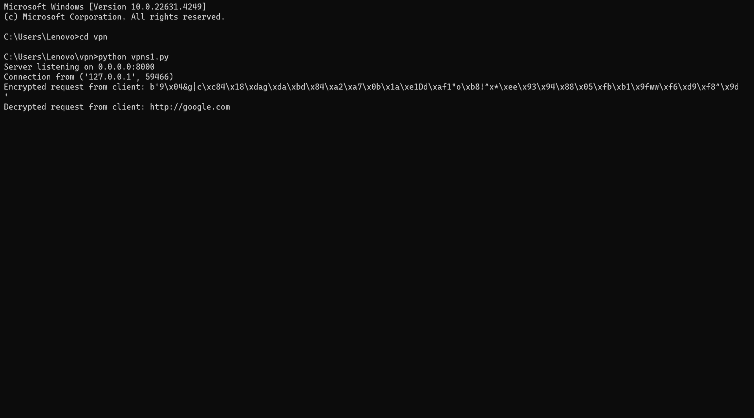
client\_socket, addr = server\_socket.accept()

handle\_client(client\_socket)

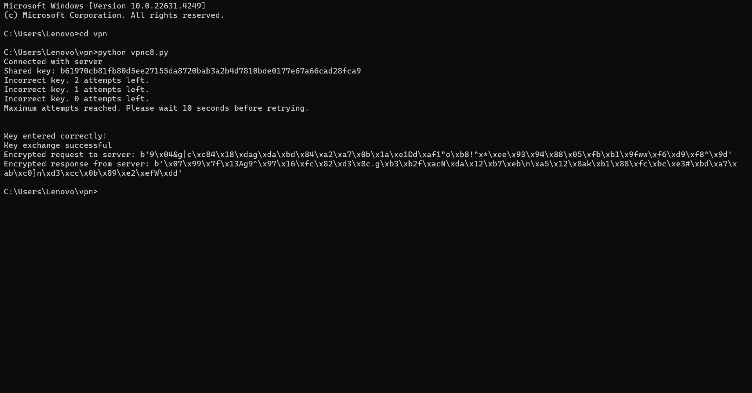
if \_\_name\_\_ == "\_\_main\_\_":

start\_server()

**OUTPUT :**



**Server Output**



**Client Output**

**RESULT:**

Thus the program for Secure VPN Communication using AES encrytion algorithm using PYTHON was verified.