**Attacking and defending networks**

**1. Encrypted covert channel**

For the exercise we have set up a server and I client which can communicate though ICMP messages.

def receive\_input():

    while True:

        print('')

        text = input("Type to send: ")

        text\_bytes = text.encode('utf-8')

        if text == 'exit()':

            os.\_exit(0)

        encrypted\_text = encrypt\_with\_aes\_cbc(preshared\_secret\_key, text\_bytes)

        send\_msg(encrypted\_text)

def send\_msg(text):

    dstip = '127.0.0.1'

    ip\_part = IP(dst=dstip)/ICMP(type=47)/Raw(load=text)

    try:

        send(ip\_part)

    except Exception as e:

        print(f"Error sending message: {e}")

def main():

    receive\_input()

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

    main()

The code above implements the client. We receive input and then encrypt it using AES in CBC mode. Then encryption is done by a preshared key. We send the ICMP msg with type 47 to 127.0.0.1.

def handle\_icmp\_message(packet):

    if ICMP in packet and Raw in packet:

        icmp\_packet = packet[ICMP]

        raw\_payload = packet[Raw].load

        if icmp\_packet.type == 47:

            # Decrypt the payload

            decrypted\_payload = decrypt\_with\_aes\_cbc(preshared\_secret\_key, raw\_payload)

            # Print the decrypted payload

            print("Received message:", decrypted\_payload.decode('utf-8'))

def main():

    # Server information

    dstip = '127.0.0.1'

    print(f"Server listening on {dstip}:")

    try:

        sniff(iface="lo", prn=handle\_icmp\_message)

    except Exception as e:

        print(f'\nError: Could not start sniffing for ICMP packets: {e}')

The code above implements the server. The server listen for ICMP packets type 47 at the loopback address 127.0.0.1. and decrypts the msg with the preshared key.

def encrypt\_with\_aes\_cbc(key, plaintext):

   # Convert hexadecimal string to bytes for the key

    key = bytes.fromhex(key)

    # Generate a random IV of the correct size (16 bytes for AES-256)

    iv = secrets.token\_bytes(16)

    # Create an AES object in CBC mode and pass the IV

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

    # Pad the plaintext to a multiple of the block size (16 bytes)

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

    # Encrypt the padded plaintext

    ciphertext = aes.encrypt(padded\_plaintext)

    # Concatenate IV and ciphertext as bytes

    encrypted\_data = iv + ciphertext

    return encrypted\_data

the encryption function is the following. We chose AES encryption for its safety, efficiency and Versatility. We chose CBC mode even though it has a few vulnerabilities. The are not implemented but could be to add MAC’s to ensure the confidentiality of the encrypted data.