**CMSC 626 Principles of Computer Security**

**Project**

**Exercise 3**

Faisal Rasheed Khan

VB02734

vb02734@umbc.edu

Shrenik PolySetty

AZ61492

1.

a. Team Information:

i. Name: **Faisal Rasheed Khan**

University Id: **VB02734**

ii. Name: **Shrenik PolySetty**

University Id: **AZ61492**

b. Secret Key: "Principles of Computer Security"

c. nc -l 12345
nc 130.85.220.34 12345
python3 rc4s.py -k 'Principles of Computer Security' -m 'Hello, it is a nice sunny day and we should enjoy the weather'
python3 rc4s.py <vm1pipe | nc 130.85.220.34 12345 >vm1pipe
nc -l 12345 <vm2pipe | python rc4r.py >vm2pipe
mkfifo vm1pipe
python3 rc4s.py -k 'Principles of Computer Security' -m 'Hello, it is a nice sunny day and we should enjoy the weather' | nc 130.85.220.34 12345
nc -l 12345 | python3 rc4r.py
python3 rc4s.py
python3 diffehellman.py
cat tcpdumpcapture.cap
nano rc4s.py
ls
ifconfig -a
tcpdump -D
tcpdump -n -i ens160 -w tcpdumpcapture.cap host 130.85.121.106 and port 12345
tcpdump -n -i ens160 -w tcpdumpcapture.cap host 130.85.220.34 and port 12345
tcpdump -n -i ens160 -w tcpdumpcapture.cap host 130.85.121.106 and port 12345
tcpdump -r capture.pcap
tcpdump -n -i ens160 -w vm1capture2.pcap host 130.85.220.34

d. The challenges faced were:

- While encrypting and decrypting the pain text and cipher text, faced issue of unable to convert bytes to text

  Resolved this issue with the help of encoding and decoding with 'utf-8'

- Faced issue while connecting to the other Virtual Machine to send the encrypted plain text using netcat command

  Resolved the issue by passing the command of netcat to subprocess.Popen()

- While encrypting and decrypting the pain text and cipher text, faced issue of unable to decode with 'utf-8' as it was in other format

  Resolved by removing unnecessary encoding.

- Faced challenges while using wireshark, tshark as access to install that was not there

  Resolved using tcpdump command

- The tcpdump command was capturing every data incoming

  Resolve by applying proper filters

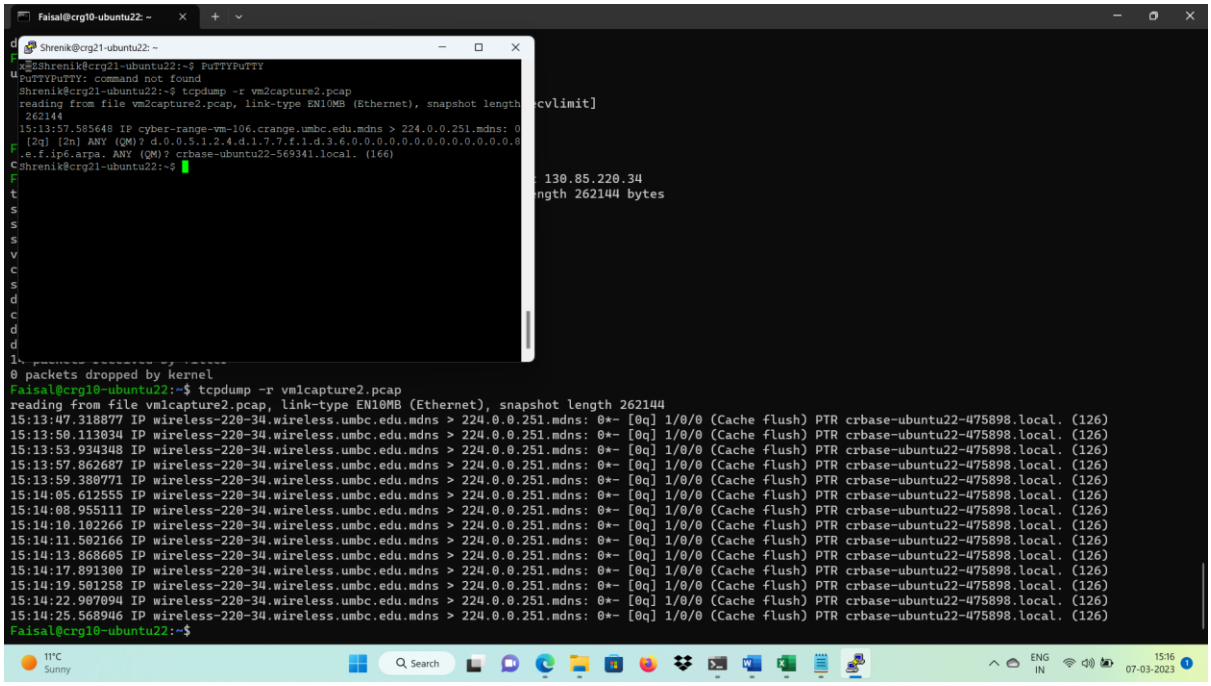
- For tcpdump command, which interface to use to capture the data was a problem

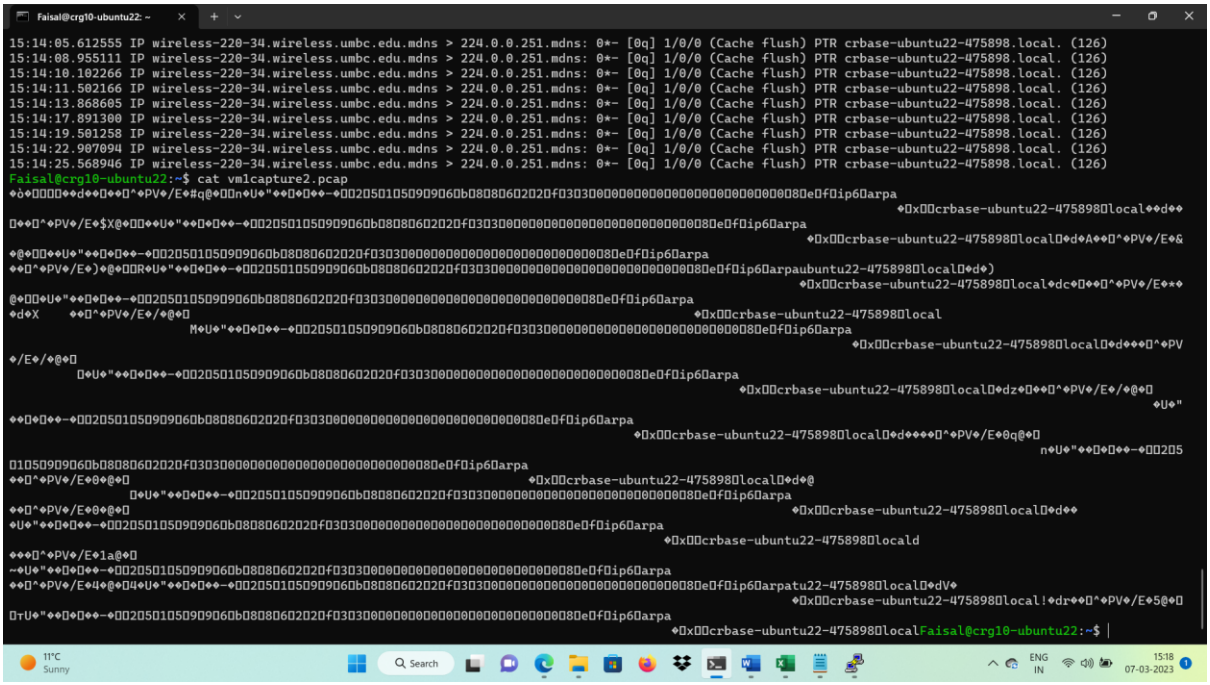
  Resolved by using this command, tcpdump -D

e. Successfully implemented the RC4 Algorithm to encrypt and decrypt the text over the communicating channels between two virtual machines. Learnt different ways of sending the data via command line arguments or incorporating everything in the python file. Successfully implemented the tcpdump/wireshark capture to capture the real time packets sent/received.

f. References:


CH02-CompSec4e_accessible_L03 (blackboardcdn.com)
How To Use Netcat to Establish and Test TCP and UDP Connections | DigitalOcean
L07-CH21-CompSec4e_accessible (blackboardcdn.com)
Primitive Root - Algorithms for Competitive Programming (cp-algorithms.com)
tcp_client.py
tcp_server.py
RC4-KeyGeneration(1).py
subprocess — Subprocess management — Python 3.11.2 documentation
Tcpdump Command in Linux | Linuxize


Faisal@crg10-ubuntu22: ~
Shrenik@crg21-ubuntu22: ~
PuTTYPuTTY: command not found
Shrenik@crg21-ubuntu22:~$ tcpdump -r vm2capture2.pcap
reading from file vm2capture2.pcap, link-type EN10MB (Ethernet), snapshot length
262144
15:13:57.585648 IP cyber-range-vm-106.crange.umbc.edu.mdns > 224.0.0.251.mdns: 0
[2q] [2n] ANY (QM)? d.0.0.5.1.2.4.d.1.7.7.f.1.d.3.6.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8
.e.f.ip6.arpa. ANY (QM)? crbase-ubuntu22-569341.local. (166)
Shrenik@crg21-ubuntu22:~$
130.85.220.34
ength 262144 bytes
0 packets dropped by kernel
Faisal@crg10-ubuntu22:~$ tcpdump -r vm1capture2.pcap
reading from file vm1capture2.pcap, link-type EN10MB (Ethernet), snapshot length 262144
15:13:47.318877 IP wireless-220-34.wireless.umbc.edu.mdns > 224.0.0.251.mdns: 0*- [0q] 1/0/0 (Cache flush) PTR crbase-ubuntu22-475898.local. (126)
15:13:50.113034 IP wireless-220-34.wireless.umbc.edu.mdns > 224.0.0.251.mdns: 0*- [0q] 1/0/0 (Cache flush) PTR crbase-ubuntu22-475898.local. (126)
15:13:53.934348 IP wireless-220-34.wireless.umbc.edu.mdns > 224.0.0.251.mdns: 0*- [0q] 1/0/0 (Cache flush) PTR crbase-ubuntu22-475898.local. (126)
15:13:57.862687 IP wireless-220-34.wireless.umbc.edu.mdns > 224.0.0.251.mdns: 0*- [0q] 1/0/0 (Cache flush) PTR crbase-ubuntu22-475898.local. (126)
15:13:59.380771 IP wireless-220-34.wireless.umbc.edu.mdns > 224.0.0.251.mdns: 0*- [0q] 1/0/0 (Cache flush) PTR crbase-ubuntu22-475898.local. (126)
15:14:05.612555 IP wireless-220-34.wireless.umbc.edu.mdns > 224.0.0.251.mdns: 0*- [0q] 1/0/0 (Cache flush) PTR crbase-ubuntu22-475898.local. (126)
15:14:08.955111 IP wireless-220-34.wireless.umbc.edu.mdns > 224.0.0.251.mdns: 0*- [0q] 1/0/0 (Cache flush) PTR crbase-ubuntu22-475898.local. (126)
15:14:10.102266 IP wireless-220-34.wireless.umbc.edu.mdns > 224.0.0.251.mdns: 0*- [0q] 1/0/0 (Cache flush) PTR crbase-ubuntu22-475898.local. (126)
15:14:11.502166 IP wireless-220-34.wireless.umbc.edu.mdns > 224.0.0.251.mdns: 0*- [0q] 1/0/0 (Cache flush) PTR crbase-ubuntu22-475898.local. (126)
15:14:13.868605 IP wireless-220-34.wireless.umbc.edu.mdns > 224.0.0.251.mdns: 0*- [0q] 1/0/0 (Cache flush) PTR crbase-ubuntu22-475898.local. (126)
15:14:17.891300 IP wireless-220-34.wireless.umbc.edu.mdns > 224.0.0.251.mdns: 0*- [0q] 1/0/0 (Cache flush) PTR crbase-ubuntu22-475898.local. (126)
15:14:19.501258 IP wireless-220-34.wireless.umbc.edu.mdns > 224.0.0.251.mdns: 0*- [0q] 1/0/0 (Cache flush) PTR crbase-ubuntu22-475898.local. (126)
15:14:22.907094 IP wireless-220-34.wireless.umbc.edu.mdns > 224.0.0.251.mdns: 0*- [0q] 1/0/0 (Cache flush) PTR crbase-ubuntu22-475898.local. (126)
15:14:25.568946 IP wireless-220-34.wireless.umbc.edu.mdns > 224.0.0.251.mdns: 0*- [0q] 1/0/0 (Cache flush) PTR crbase-ubuntu22-475898.local. (126)
Faisal@crg10-ubuntu22:~$
11°C
Sunny
Search
ENG
IN
15:16
07-03-2023


Faisal@crg10-ubuntu22: ~
15:14:05.612555 IP wireless-220-34.wireless.umbc.edu.mdns > 224.0.0.251.mdns: 0*- [0q] 1/0/0 (Cache flush) PTR crbase-ubuntu22-475898.local. (126)
15:14:08.955111 IP wireless-220-34.wireless.umbc.edu.mdns > 224.0.0.251.mdns: 0*- [0q] 1/0/0 (Cache flush) PTR crbase-ubuntu22-475898.local. (126)
15:14:10.102266 IP wireless-220-34.wireless.umbc.edu.mdns > 224.0.0.251.mdns: 0*- [0q] 1/0/0 (Cache flush) PTR crbase-ubuntu22-475898.local. (126)
15:14:11.502166 IP wireless-220-34.wireless.umbc.edu.mdns > 224.0.0.251.mdns: 0*- [0q] 1/0/0 (Cache flush) PTR crbase-ubuntu22-475898.local. (126)
15:14:13.868605 IP wireless-220-34.wireless.umbc.edu.mdns > 224.0.0.251.mdns: 0*- [0q] 1/0/0 (Cache flush) PTR crbase-ubuntu22-475898.local. (126)
15:14:17.891300 IP wireless-220-34.wireless.umbc.edu.mdns > 224.0.0.251.mdns: 0*- [0q] 1/0/0 (Cache flush) PTR crbase-ubuntu22-475898.local. (126)
15:14:19.501258 IP wireless-220-34.wireless.umbc.edu.mdns > 224.0.0.251.mdns: 0*- [0q] 1/0/0 (Cache flush) PTR crbase-ubuntu22-475898.local. (126)
15:14:22.907094 IP wireless-220-34.wireless.umbc.edu.mdns > 224.0.0.251.mdns: 0*- [0q] 1/0/0 (Cache flush) PTR crbase-ubuntu22-475898.local. (126)
15:14:25.568946 IP wireless-220-34.wireless.umbc.edu.mdns > 224.0.0.251.mdns: 0*- [0q] 1/0/0 (Cache flush) PTR crbase-ubuntu22-475898.local. (126)
Faisal@crg10-ubuntu22:~$ cat vm1capture2.pcap
11°C
Sunny
Search
ENG
IN
15:18
07-03-2023

2.

**Code on VM1(Faisal Rasheed Khan)** 130.85.121.106**:**

**rc4.py:**

```
def KSA(key):
    #Key Scheduling Algorithm (KSA) for RC4.
    Keylength = len(key)

    # Initialize permutation array
    S = list(range(256))

    # Use key to permute the array
    j = 0
    for i in range(256):
        j = (j + S[i] + key[i % keylength]) % 256
        S[i], S[j] = S[j], S[i]  # Swap values

    return S

def PRGA(S, length):
    #Pseudo-Random Generation Algorithm (PRGA) for RC4.
    I = j = 0
    keystream = []
    for _ in range(length):
        i = (i + 1) % 256
        j = (j + S[i]) % 256
        S[i], S[j] = S[j], S[i]  # Swap values
        k = S[(S[i] + S[j]) % 256]
        keystream.append(k)

    return keystream
```

```
def encrypt(key, data):
    #RC4 encryption  algorithm.
    S = KSA(key)
    keystream = PRGA(S, len(data))
    output = [data[i] ^ keystream[i] for i in range(len(data))]
    return str(output)

def decrypt(key, data):
    #RC4 decryption algorithm.
    S = KSA(key)
    keystream = PRGA(S, len(data))
    output = [data[i] ^ keystream[i] for i in range(len(data))]
    return bytes(output)
key = b”Principles of Computer Security”
```

**rc4s.py:**

```
import argparse
import subprocess
import signal
import rc4
import socket
import time
'''
import subprocess

# Create the named pipe
subprocess.call('mkfifo vm1pipe', shell=True)

# Open the pipe for writing
```

```
pipe = open('vm1pipe', 'w')

# Start the nc command to listen on port 12345 and write to the pipe
#nc_proc
NC = subprocess.Popen(['nc', '130.85.220.34', '12345'], stdout=pipe, stderr=subprocess.STDOUT)

# Wait for 1 second to make sure nc is fully established
time.sleep(1)

# Open the pipe for reading and pass it to rc4s
with open('vm1pipe', 'r') as input_pipe:
    subprocess.call(['python3', 'rc4s.py'], stdin=input_pipe)

# Close the pipe and terminate the nc process
pipe.close()
NC.terminate()

parser = argparse.ArgumentParser(description='Encrypt and send message using RC4')
parser.add_argument('-k', '--key', type=str, required=True, help='key for RC4 encryption')
parser.add_argument('-m', '--message', type=str, required=True, help='message to be encrypted')
#parser.add_argument('ip', type=str, help='IP address of receiver')
#parser.add_argument('port', type=int, help='port of receiver')
args = parser.parse_args()
rc4.key=args.key
en_messageargs.message
'''
# the netcat command to run and redirect its input/output to pipes
#pipelining the 2 VM inorder to encrypt and decrypt  using shared key
os_cmd = ['nc', '130.85.220.34', '12345']

#NetCat
```

```
NC = subprocess.Popen(os_cmd, stdin=subprocess.PIPE, stdout=subprocess.PIPE)


print("Shared Key : " + rc4.key.decode("utf-8"))


tcpdump_cmd1 = ['tcpdump',  '-i', 'ens160', '-s','0','w','capture.pcap','&', 'host', '130.85.121.106',
'and', '130.85.220.34']

# to store the tcpdump in file

with open('tcpdump3.txt', 'w') as outfile:

   tcpdump_proc = subprocess.Popen(tcpdump_cmd1, stdout=outfile, stderr=subprocess.STDOUT)


# tcpdump capture / wireshark capture, can use either .cap or .pcap

tcpdump_cmd = ['tcpdump', '-n', '-i', 'ens160', '-w','tcpdumpcapture.cap', 'host', '130.85.121.106',
'and', 'port', '12345']


# to store the packets of wireshark capture in .txt file

with open('tcpdump1.txt', 'w') as outfile:

   tcpdump_proc = subprocess.Popen(tcpdump_cmd, stdout=outfile, stderr=subprocess.STDOUT)


en_message=bytes('Hello, it is a nice sunny day and we should enjoy the weather','utf-8')

#Encrypt the plain text to send to the receiver

encrypted_message = rc4.encrypt(rc4.key,en_message)

'''with open("vm2pipe", "w") as pipe:

      pipe.write(encrypted_message.encode().decode())

'''

NC.stdin.write(encrypted_message.encode() + b'\n')

NC.stdin.flush()

'''

srvr_addr = ('130.85.220.34', 9999)

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

sock.connect(srvr_addr)

sock.delay(5)

sent = sock.send(encrypted_message.encode())
```

```
#sent = sock.send(msg.encode())
time.sleep(1)
#sock.close()
'''
#tcpdump_proc.send_signal(signal.SIGINT)
# receive the encrypted message from the receiver using netcat
#received_message = subprocess.check_output(nc_cmd)
'''with open("vm2pipe", "r") as pipe:
      cipher = pipe.read().strip()
'''
cipher = NC.stdout.readline().decode().strip()
'''
sock.bind(srvr_addr)
sock.listen(5)
connsock, client = sock.accept()
cipher = connsock.recv(20)
'''
cipher=cipher[1:-1].split(", ")
cipher=[eval(i) for i in cipher]
cipher=bytes(cipher)

decrypted_message = rc4.decrypt(rc4.key,cipher)
#print the decrypted message
print(decrypted_message.decode("utf-8"))

#sock.close()
tcpdump_proc.send_signal(signal.SIGINT)
```

**Code on VM2(Shrenik PolySetty)** 130.85.220.34**:**

**rc4.py:**

```
def KSA(key):
    #Key Scheduling Algorithm (KSA) for RC4.
    Keylength = len(key)

    # Initialize permutation array
    S = list(range(256))

    # Use key to permute the array
    j = 0
    for i in range(256):
        j = (j + S[i] + key[i % keylength]) % 256
        S[i], S[j] = S[j], S[i]  # Swap values

    return S

def PRGA(S, length):
    #Pseudo-Random Generation Algorithm (PRGA) for RC4.
    I = j = 0
    keystream = []
    for _ in range(length):
        i = (i + 1) % 256
        j = (j + S[i]) % 256
        S[i], S[j] = S[j], S[i]  # Swap values
        k = S[(S[i] + S[j]) % 256]
        keystream.append(k)

    return keystream

def encrypt(key, data):
    #RC4 encryption  algorithm.
```

```
    S = KSA(key)
    keystream = PRGA(S, len(data))
    output = [data[i] ^ keystream[i] for i in range(len(data))]
    return str(output)

def decrypt(key, data):
    #RC4 decryption algorithm.
    S = KSA(key)
    keystream = PRGA(S, len(data))
    output = [data[i] ^ keystream[i] for i in range(len(data))]
    return bytes(output)
key = b”Principles of Computer Security”
```

**rc4r.py:**

```
import argparse
import subprocess
import signal
import rc4
#from rc4 import RC4

'''parser = argparse.ArgumentParser(description='Receive and decrypt message using RC4')
#parser.add_argument('port', type=int, help='port to listen on')
parser.add_argument('-k', '--key', type=str, required=True, help='key for RC4 encryption')
#parser.add_argument('port', type=int, help='port to listen on')
args = parser.parse_args()'''
os_cmd = ['nc', '-l', '12345']
NC = subprocess.Popen(os_cmd, stdin=subprocess.PIPE, stdout=subprocess.PIPE)

#with open("vm1pipe","r") as pipe:
#    B=pipe.read().strip()
```

```
'''
rc4.key=NC.stdout.readline().decode("utf-8").strip()
print("Shared Key :")
print(rc4.key)
rc4.key=bytes(rc4.key,"utf-8")
'''
print("Shared key : "+rc4.key.decode("utf-8"))


tcpdump_cmd1 = ['tcpdump',  '-i', 'ens160', '-s','0','w','capture.pcap','&', 'host', '130.85.220.34',
'and', '130.85.121.106']
# to store the tcpdump in file
with open('tcpdump3.txt', 'w') as outfile:
    tcpdump_proc = subprocess.Popen(tcpdump_cmd1, stdout=outfile, stderr=subprocess.STDOUT)


cipher=NC.stdout.readline().decode("utf-8").strip()
cipher=cipher[1:-1].split(", ")
cipher=[eval(i) for i in cipher]
cipher=bytes(cipher)


decrypted_message = rc4.decrypt(rc4.key,cipher)
# print the decrypted message
print(decrypted_message.decode("utf-8"))


# encrypt the response message using RC4


response_message = 'Busy with assignment right now.'
#response_message = b'Busy with assignment right now.'
encrypted_response = rc4.encrypt(rc4.key,bytes(response_message,'utf-8'))


tcpdump_cmd = ['tcpdump', '-n', '-i', 'ens160', '-w','tcpdumpcapture.cap', 'host', '130.85.220.34',
'and', 'port', '12345']
```

```
# to store the tcpdump in file
with open('tcpdump1.txt', 'w') as outfile:
    tcpdump_proc = subprocess.Popen(tcpdump_cmd, stdout=outfile, stderr=subprocess.STDOUT)



#with open("vm1pipe","w") as pipe:

NC.stdin.write(str(encrypted_response).encode("utf-8")+ b'\n')
NC.stdin.flush()
tcpdump_proc.send_signal(signal.SIGINT)

'''import sys
import subprocess

# read the encrypted message from netcat
encrypted_message = sys.stdin.buffer.read()

# decrypt the message using RC4
cipher = RC4(args.key.encode())
decrypted_message = cipher.decrypt(encrypted_message)

# display the decrypted message
print(decrypted_message.decode())

# send the decrypted message back to sender using netcat
nc_cmd = f'nc {args.ip} {args.port} >{sys.stdout.fileno()}'
subprocess.run(nc_cmd, input=decrypted_message, shell=True)
'''
```

**Exercise 4**

Faisal Rasheed Khan

VB02734

vb02734@umbc.edu

Shrenik PolySetty

AZ61492

1.

a. Team Information:

i. Name: **Faisal Rasheed Khan**

University Id: **VB02734**

ii. Name: **Shrenik PolySetty**

University Id: **AZ61492**

b. Prime Number, P = 1065601
Primitive root, α = 7
Xa = 139278
Xb = 111689
Key = 159571

c. nc -l 12345
nc 130.85.220.34 12345
python3 diffehellman.py | nc 130.85.220.34 12345
python3 diffehellman.py <vm1pipe | nc 130.85.220.34 12345 >vm1pipe
nc -l 12345 <vm2pipe | python diffehellman.py >vm2pipe
mkfifo vm1pipe
nc -l 12345 | python3 diffehellman.py
python3 diffehellman.py
cat tcpdumpcapture.cap
nano diffehellman.py
ls
ifconfig -a
tcpdump -D
tcpdump -n -i ens160 -w tcpdumpcapture.cap host 130.85.121.106 and port 12345
tcpdump -n -i ens160 -w tcpdumpcapture.cap host 130.85.220.34 and port 12345
tcpdump -n -i ens160 -w tcpdumpcapture.cap host 130.85.121.106 and port 12345
tcpdump -r capture.pcap
tcpdump -n -i ens160 -w vm1capture2.pcap host 130.85.220.34

d. The challenges faced were:
- While encrypting and decrypting the key, faced issue because key was of int type text

Resolved this issue with the help of converting the int to str ,encoding and decoding with 'utf-8'

- Faced issue while connecting to the other Virtual Machine to send the encrypted plain text using netcat command

  Resolved the issue by passing the command of netcat to subprocess.Popen()

- While encrypting and decrypting the pain text and cipher text, faced issue of unable to decode with 'utf-8' as it was in other format

  Resolved by removing unnecessary encoding.

- Faced challenges while using wireshark, tshark as access to install that was not there

  Resolved using tcpdump command

- The tcpdump command was capturing every data incoming

  Resolve by applying proper filters

- For tcpdump command, which interface to use to capture the data was a problem

  Resolved by using this command, tcpdump -D

e. Successfully implemented the Diffe-Hellman Key Exchange Algorithm to encrypt and decrypt the text over the communicating channels between two virtual machines while calculating the shared key with their respective private keys, where keys are generated using prime number, primitive root and private key. Learnt different ways of sending the data via command line arguments or incorporating everything in the python file. Successfully implemented the tcpdump/wireshark capture to capture the real time packets sent/received. Can store tcpdump/wireshark capture using .pcap or .cap extension.

f. References:


CH02-CompSec4e_accessible_L03 (blackboardcdn.com)

How To Use Netcat to Establish and Test TCP and UDP Connections | DigitalOcean

L07-CH21-CompSec4e_accessible (blackboardcdn.com)

Primitive Root - Algorithms for Competitive Programming (cp-algorithms.com)

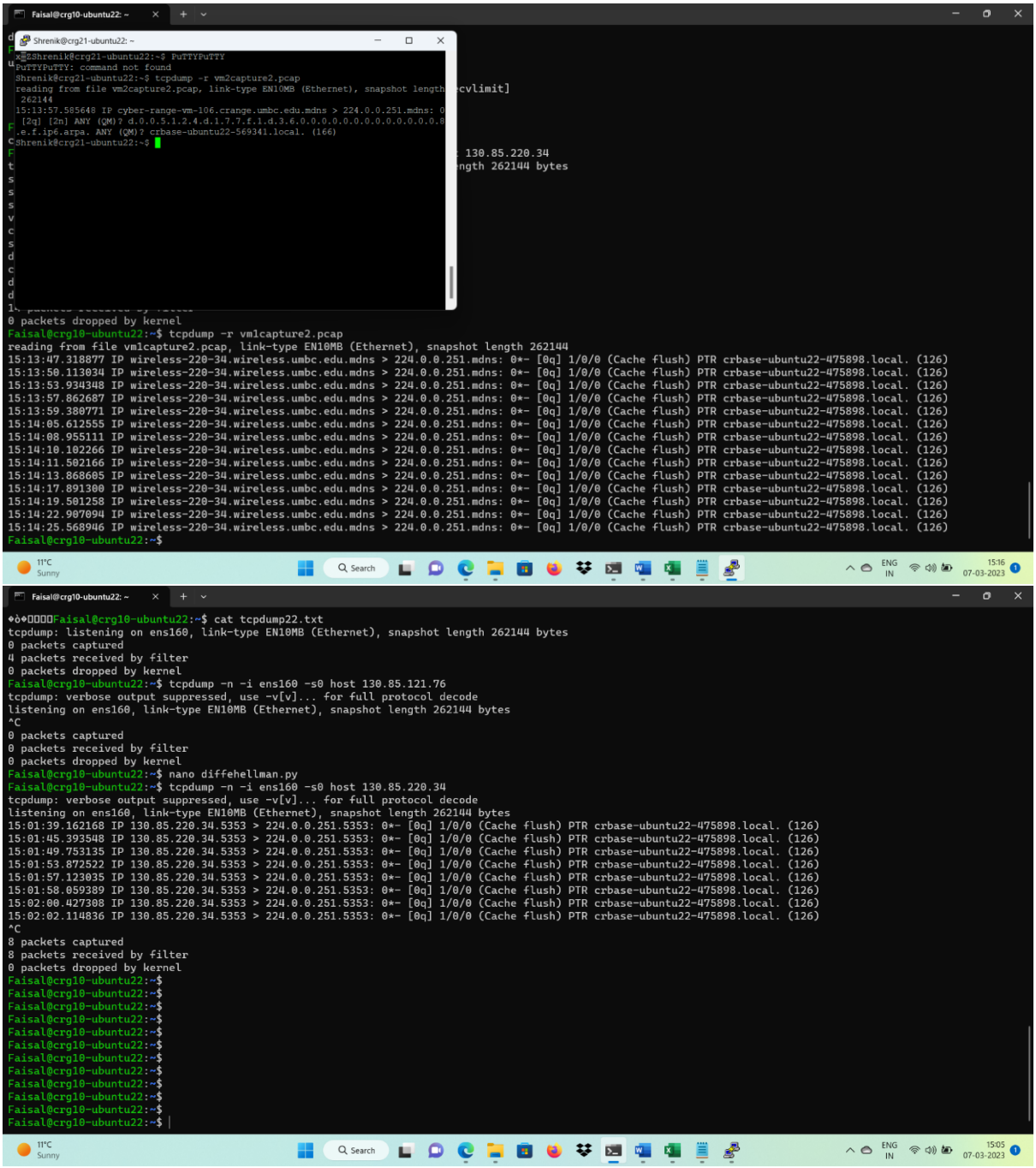
tcp_client.py

tcp_server.py

RC4-KeyGeneration(1).py

subprocess — Subprocess management — Python 3.11.2 documentation

Tcpdump Command in Linux | Linuxize




2.

**Code on VM1(Faisal Rasheed Khan)** 130.85.121.106**:**

**rc4.py:**

```
def KSA(key):
    #Key Scheduling Algorithm (KSA) for RC4.
```

```
    Keylength = len(key)

    # Initialize permutation array
    S = list(range(256))

    # Use key to permute the array
    j = 0
    for i in range(256):
        j = (j + S[i] + key[i % keylength]) % 256
        S[i], S[j] = S[j], S[i]  # Swap values

    return S

def PRGA(S, length):
    #Pseudo-Random Generation Algorithm (PRGA) for RC4.
    I = j = 0
    keystream = []
    for _ in range(length):
        i = (i + 1) % 256
        j = (j + S[i]) % 256
        S[i], S[j] = S[j], S[i]  # Swap values
        k = S[(S[i] + S[j]) % 256]
        keystream.append(k)

    return keystream

def encrypt(key, data):
    #RC4 encryption  algorithm.
    S = KSA(key)
    keystream = PRGA(S, len(data))
    output = [data[i] ^ keystream[i] for i in range(len(data))]
```

```
    return str(output)

def decrypt(key, data):
    #RC4 decryption algorithm.
    S = KSA(key)
    keystream = PRGA(S, len(data))
    output = [data[i] ^ keystream[i] for i in range(len(data))]
    return bytes(output)
```

**diffehellman.py:**

```
import subprocess
import random
import signal
import rc4
# the prime and primitive root.
p = 1065601
# took the value by finding the value from the function defined below
g = 7
def is_primitive_root(g, p):
    """
    Check if g is a primitive root of prime p
    """
    s = set(pow(g, i, p) for i in range(1, p))
    return len(s) == p-1

def find_primitive_root(p):
    """
    Find a primitive root of prime p
    """
    for g in range(2, p):
```

```
        if is_primitive_root(g, p):
            return g
    return None
#gg = find_primitive_root(p)
#print(gg)

# Generate a random private key
private_key = random.randint(1000, p-2)
print("Private Key YA :",private_key)
# the netcat command to run and redirect its input/output to pipes
os_cmd = ['nc', '130.85.220.34', '12345']
NC = subprocess.Popen(os_cmd, stdin=subprocess.PIPE, stdout=subprocess.PIPE)

interface = 'ens160'
ip_address = '130.85.220.34'
output_file = 'sender.pcap'
#tcpdump_cmd1 = f'tcpdump -i {interface} src {ip_address} -w {output_file}'
#tcpdump_cmd2= f'tcpdump -i ens160 -s 0 -w capture.pcap &'
tcpdump_cmd2= ['tcpdump', '-i', 'ens160', '-s', '0', '-w' ,'capture.pcap', '&','host', '130.85.220.34']
#subprocess.run(tcpdump_cmd2, shell=True)

with open('tcpdump3.txt', 'w') as outfile:
    tcpdump_proc = subprocess.Popen(tcpdump_cmd2, stdout=outfile, stderr=subprocess.STDOUT)
tcpdump_cmd = ['tcpdump', '-n', '-i', 'ens160', '-w','vm1capture.pcap', 'host', '130.85.121.106', 'and',
'port', '12345']

# to store the tcpdump in file
with open('tcpdump2.txt', 'w') as outfile:
    tcpdump_proc = subprocess.Popen(tcpdump_cmd, stdout=outfile, stderr=subprocess.STDOUT)

#Calculate YA = g^private_key mod p
```

```
YA = pow(g, private_key, p)

# Send YA to receiver
NC.stdin.write(str(YA).encode("utf-8") + b'\n')
NC.stdin.flush()
# Receive YA from sender
YB = int(NC.stdout.readline().decode("utf-8").strip())
# Calculate the shared secret key K =  YA^private_key mod p
K = pow(YB, private_key, p)

print('Shared Secret Key:', K)

key = bytes(str(K), 'utf-8')

#send message to VM2
#send=input("Send message : ")
send="Hello, it is a nice sunny day and we should enjoy the weather"
plaintext = bytes(send, 'utf-8')
ciphertext_encrypt = rc4.encrypt(key, plaintext)
#ciphertext_encrypt = RC4_encryption(key, plaintext)
NC.stdin.write(ciphertext_encrypt.encode("utf-8") + b'\n')
NC.stdin.flush()


# received message from VM2
cipher_decrypt = NC.stdout.readline().decode("utf-8").strip()
cipher_decrypt=cipher_decrypt[1:-1].split(", ")
cipher_decrypt = [eval(i) for i in cipher_decrypt]
cipher_decrypt = bytes(cipher_decrypt)
#,"utf-8")
plain_decrypt = rc4.decrypt(key, cipher_decrypt)
```

```
#decrypted = RC4_decryption(key, response)
print("Received message :", plain_decrypt.decode("utf-8"))
tcpdump_proc.send_signal(signal.SIGINT)
```

**Code on VM2(Shrenik PolySetty)** 130.85.220.34**:**

**rc4.py:**

```
def KSA(key):
    #Key Scheduling Algorithm (KSA) for RC4.
    Keylength = len(key)

    # Initialize permutation array
    S = list(range(256))

    # Use key to permute the array
    j = 0
    for i in range(256):
        j = (j + S[i] + key[i % keylength]) % 256
        S[i], S[j] = S[j], S[i]  # Swap values

    return S

def PRGA(S, length):
    #Pseudo-Random Generation Algorithm (PRGA) for RC4.
    I = j = 0
    keystream = []
    for _ in range(length):
        i = (i + 1) % 256
        j = (j + S[i]) % 256
        S[i], S[j] = S[j], S[i]  # Swap values
        k = S[(S[i] + S[j]) % 256]
```

```
        keystream.append(k)

    return keystream

def encrypt(key, data):
    #RC4 encryption  algorithm.
    S = KSA(key)
    keystream = PRGA(S, len(data))
    output = [data[i] ^ keystream[i] for i in range(len(data))]
    return str(output)

def decrypt(key, data):
    #RC4 decryption algorithm.
    S = KSA(key)
    keystream = PRGA(S, len(data))
    output = [data[i] ^ keystream[i] for i in range(len(data))]
    return bytes(output)
```

**diffehellman.py:**

```
import subprocess
import random
import signal
import rc4
# the prime and primitive root.
p = 1065601
g = 7

# Generate a random private key
private_key = random.randint(1000, p-2)
print("private key YB:",private_key)
```

```
# the netcat command to run and redirect its input/output to pipes
os_cmd = ['nc', '-l', '12345']
NC = subprocess.Popen(os_cmd, stdin=subprocess.PIPE, stdout=subprocess.PIPE)



tcpdump_cmd1 = ['tcpdump',  '-i', 'ens160', '-s','0','w','capture.pcap','&', 'host', '130.85.220.34',
'and', '130.85.121.106']


tcpdump_cmd = ['tcpdump', '-n', '-i', 'ens160', '-w','vm1capture.pcap', 'host', '130.85.220.34', 'and',
'port', '12345']


# to store the tcpdump in file
with open('tcpdump3.txt', 'w') as outfile:
    tcpdump_proc = subprocess.Popen(tcpdump_cmd1, stdout=outfile, stderr=subprocess.STDOUT)
# to store the tcpdump in file
with open('tcpdump2.txt', 'w') as outfile:
    tcpdump_proc = subprocess.Popen(tcpdump_cmd, stdout=outfile, stderr=subprocess.STDOUT)


# Receive YA from computer A
YA = int(NC.stdout.readline().decode("utf-8").strip())


# Calculate YB = g^private_key mod p
YB = pow(g, private_key, p)


# Send YB to computer A
NC.stdin.write(str(YB).encode("utf-8") + b'\n')
NC.stdin.flush()


# Calculate the shared secret key K =  YA^private_key mod p
K = pow(YA, private_key, p)
```

```
print('Shared Secret Key:', K)

key = bytes(str(K), 'utf-8')

#received message from VM2
cipher_decrypt = NC.stdout.readline().decode("utf-8").strip()
cipher_decrypt=cipher_decrypt[1:-1].split(", ")
cipher_decrypt = [eval(i) for i in cipher_decrypt]
cipher_decrypt = bytes(cipher_decrypt)
#,"utf-8")
plain_decrypt = rc4.decrypt(key, cipher_decrypt)
print("Received message :", plain_decrypt.decode("utf-8"))

#send message to VM2
#send=input("Send message : ")
send="Busy with assignment right now. "
plaintext = bytes(send, 'utf-8')
ciphertext_encrypt = rc4.encrypt(key, plaintext)
#ciphertext_encrypt = RC4_encryption(key, plaintext)
NC.stdin.write(ciphertext_encrypt.encode("utf-8") + b'\n')
NC.stdin.flush()

tcpdump_proc.send_signal(signal.SIGINT)
```

References:


CH02-CompSec4e_accessible_L03 (blackboardcdn.com)

How To Use Netcat to Establish and Test TCP and UDP Connections | DigitalOcean

L07-CH21-CompSec4e_accessible (blackboardcdn.com)

Primitive Root - Algorithms for Competitive Programming (cp-algorithms.com)

tcp_client.py

tcp_server.py

RC4-KeyGeneration(1).py

subprocess — Subprocess management — Python 3.11.2 documentation

Tcpdump Command in Linux | Linuxize