

Olban Abraham Lopez Aranda

Ian Thomas Burres

Network Security / IS-3423

October 12 of the 2025

## Lab 2 Report: Huffman, Capacity, and Data Flows

Part 1-

Problem A – Huffman Encoding for “bookkeeper”

1- Count frequencies:

Letter	Count
b	1
o	2
k	2
e	3
p	1
r	1
Total	10

2- Assign Huffman codes (one possible valid set):

Letter	Frequency	Code	Length
e	3	0	1
k	2	10	2
o	2	11	2
b	1	100	3
p	1	101	3
r	1	110	3

3- Compute total bits:

$$(3 \times 1) + (2 \times 2) + (2 \times 2) + (1 \times 3) + (1 \times 3) + (1 \times 3) = 20 \text{ bits}$$

4- Average bits per character:

$$20 \text{ bits} \div 10 \text{ characters} = 2$$

### Problem B- Decode the Bitsring

b → 00
o → 01
k → 10
e → 110
p → 1110
r → 1111

Provided bitstring (incorrect example):

10100110101100111000101111110

This does not decode correctly to bookkeeper because pieces like 101 and 100 don't match any valid mapping.

Correct encoding for “bookkeeper”:

b → 00
o → 01
o → 01
k → 10
k → 10
e → 110
e → 110
p → 1110
e → 110
r → 1111

Delimited version:

00 01 01 10 10 110 110 1110 110 1111

Concatenated bitstring:

000101101011011011101101111

### Problem C – Huffman vs Fixed-Length Encoding

- Distinct letters = 6 (b, o, k, e, p, r)
- Fixed-length bits per symbol =  $\lceil \log_2 6 \rceil = 3$  bits
- Total fixed-length bits =  $10 \times 3 = 30$  bits
- Huffman bits (from Problem A) = 20 bits
- Bits saved =  $30 - 20 = 10$  bits

### Problem D – Channel Capacity of a Binary Symmetric Channel

Formula

$$C = 1 - H(p)$$

where

$$H(p) = -p \log_2 p - (1-p) \log_2 (1-p)$$

Substitute p = 0.1:

$$\begin{aligned} H(0.1) &= -(0.1) \log_2 (0.1) - (0.9) \log_2 (0.9) \\ C &= 1 - [-(0.1) \log_2 (0.1) - (0.9) \log_2 (0.9)] \end{aligned}$$

Approximation:

$$H(0.1) \approx 0.469 \rightarrow C \approx 0.531 \text{ bits per bit sent}$$

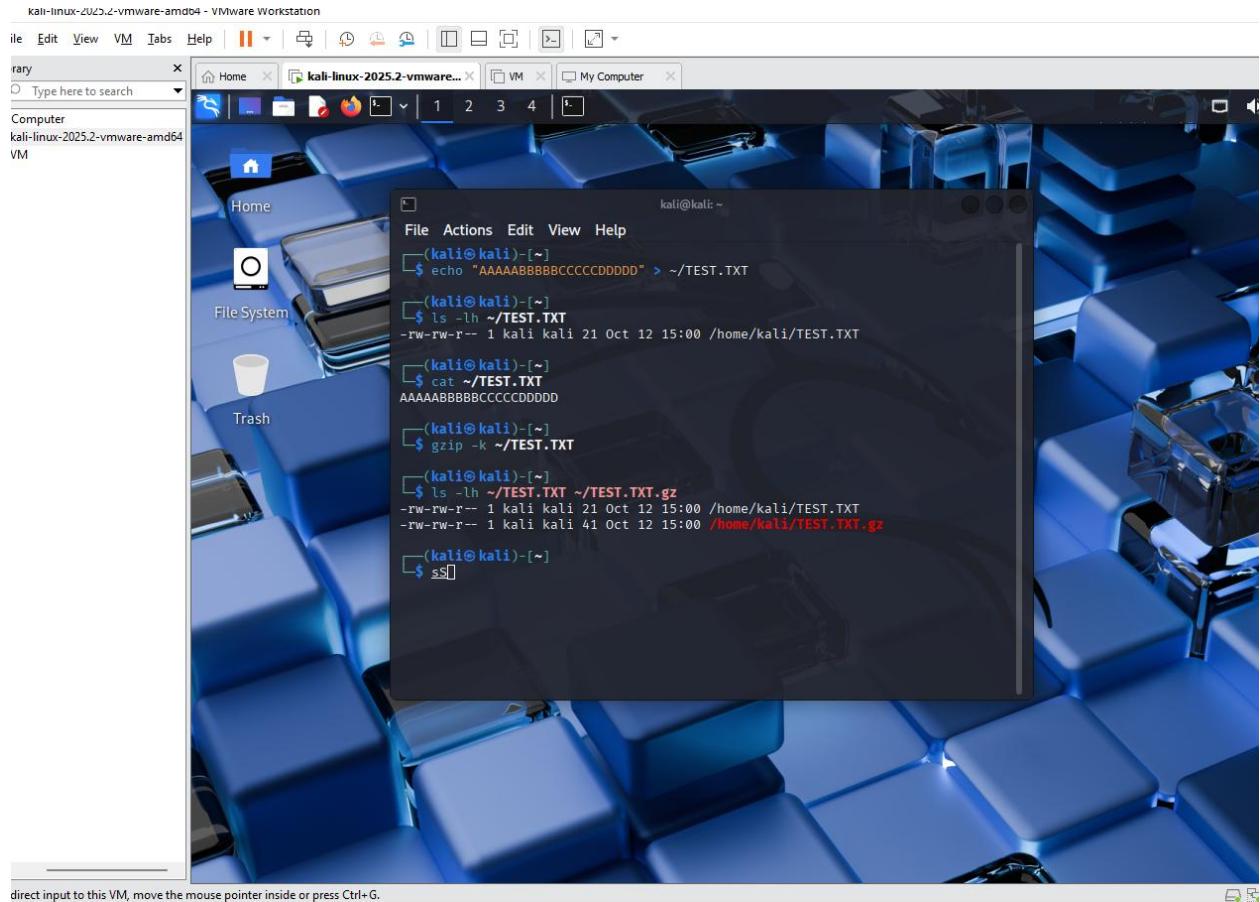
Interpretation:

When p = 0.1, the channel can reliably carry about 53.1 % of its ideal bit rate.

As error probability p increases, capacity drops because more bits must be used for error correction.

## Part 2 of the lab

### Step 1- Compression Demo



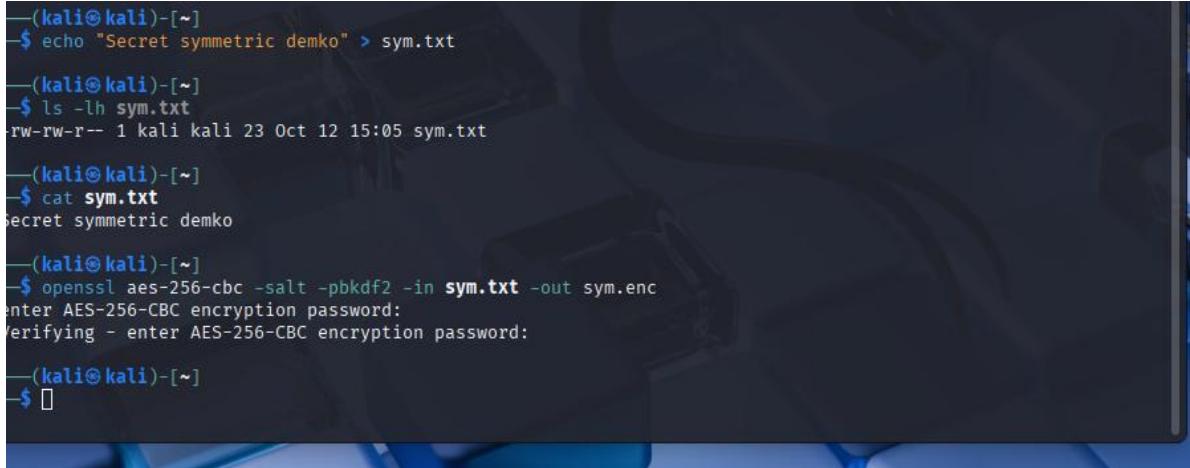
The screenshot shows a VMware Workstation interface with a Kali Linux 2025.2 VM running. A terminal window is open, displaying the following command sequence:

```
(kali㉿kali)-[~]
$ echo "AAAAABBBBBCCCCDDDD" > ~/TEST.TXT
(kali㉿kali)-[~]
$ ls -lh ~/TEST.TXT
-rw-rw-r-- 1 kali kali 21 Oct 12 15:00 /home/kali/TEST.TXT
(kali㉿kali)-[~]
$ cat ~/TEST.TXT
AAAAABBBBBCCCCDDDD
(kali㉿kali)-[~]
$ gzip -k ~/TEST.TXT
(kali㉿kali)-[~]
$ ls -lh ~/TEST.TXT ~/TEST.TXT.gz
-rw-rw-r-- 1 kali kali 21 Oct 12 15:00 /home/kali/TEST.TXT
-rw-rw-r-- 1 kali kali 41 Oct 12 15:00 /home/kali/TEST.TXT.gz
(kali㉿kali)-[~]
$ ss[
```

A message at the bottom of the screen reads: "direct input to this VM, move the mouse pointer inside or press Ctrl+G."

*Screenshot showing the creation of “test.txt”, use of the gzip command, and file size comparison between the original and compressed versions.*

## Step 2



```
(kali㉿kali)-[~]
$ echo "Secret symmetric demko" > sym.txt

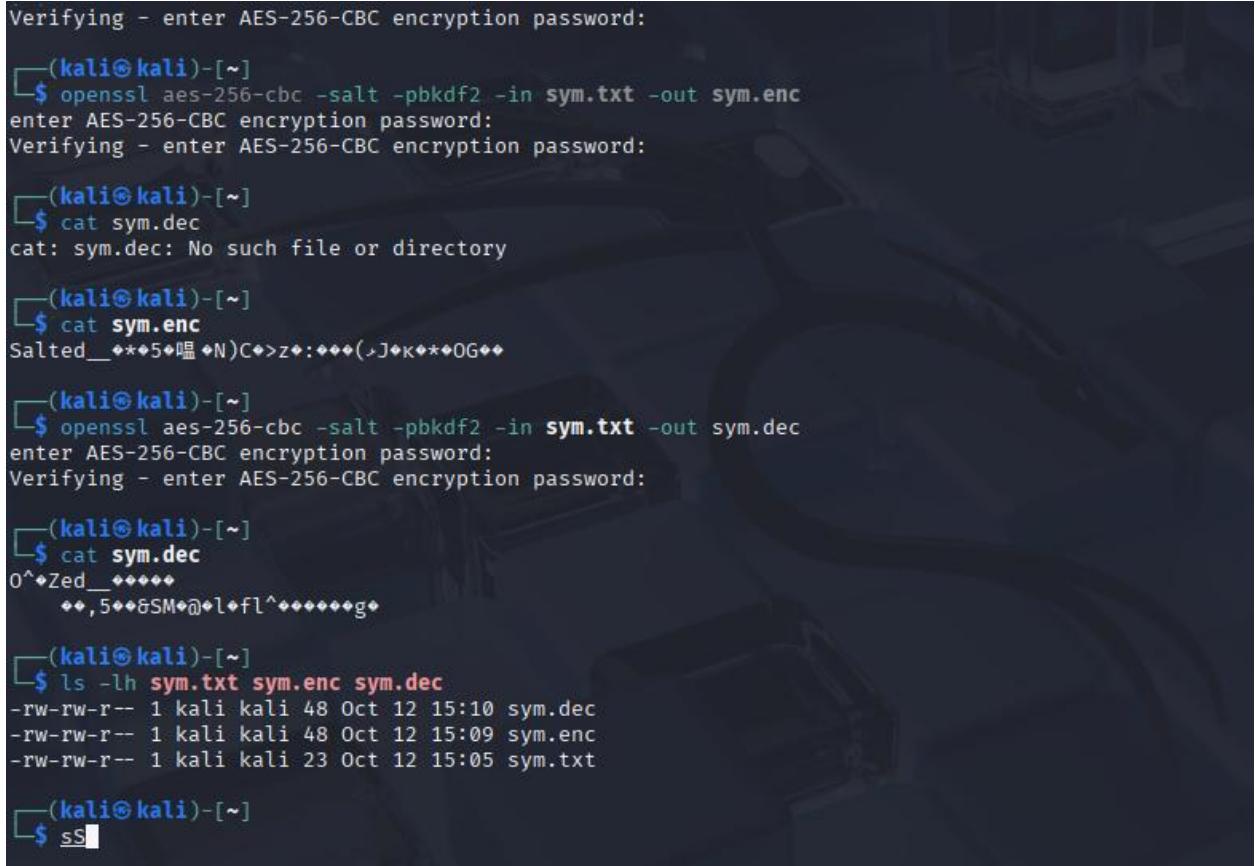
(kali㉿kali)-[~]
$ ls -lh sym.txt
-rw-rw-r-- 1 kali kali 23 Oct 12 15:05 sym.txt

(kali㉿kali)-[~]
$ cat sym.txt
Secret symmetric demko

(kali㉿kali)-[~]
$ openssl aes-256-cbc -salt -pbkdf2 -in sym.txt -out sym.enc
enter AES-256-CBC encryption password:
Verifying - enter AES-256-CBC encryption password:

(kali㉿kali)-[~]
$ 
```

Screenshot showing AES-256-CBC encryption command with password prompt used to secure “sym.txt”.



```
Verifying - enter AES-256-CBC encryption password:
(kali㉿kali)-[~]
$ openssl aes-256-cbc -salt -pbkdf2 -in sym.txt -out sym.enc
enter AES-256-CBC encryption password:
Verifying - enter AES-256-CBC encryption password:

(kali㉿kali)-[~]
$ cat sym.dec
cat: sym.dec: No such file or directory

(kali㉿kali)-[~]
$ cat sym.enc
Salted__***5*囧+N)C>z*:***(>J*k**OG**

(kali㉿kali)-[~]
$ openssl aes-256-cbc -salt -pbkdf2 -in sym.txt -out sym.dec
enter AES-256-CBC encryption password:
Verifying - enter AES-256-CBC encryption password:

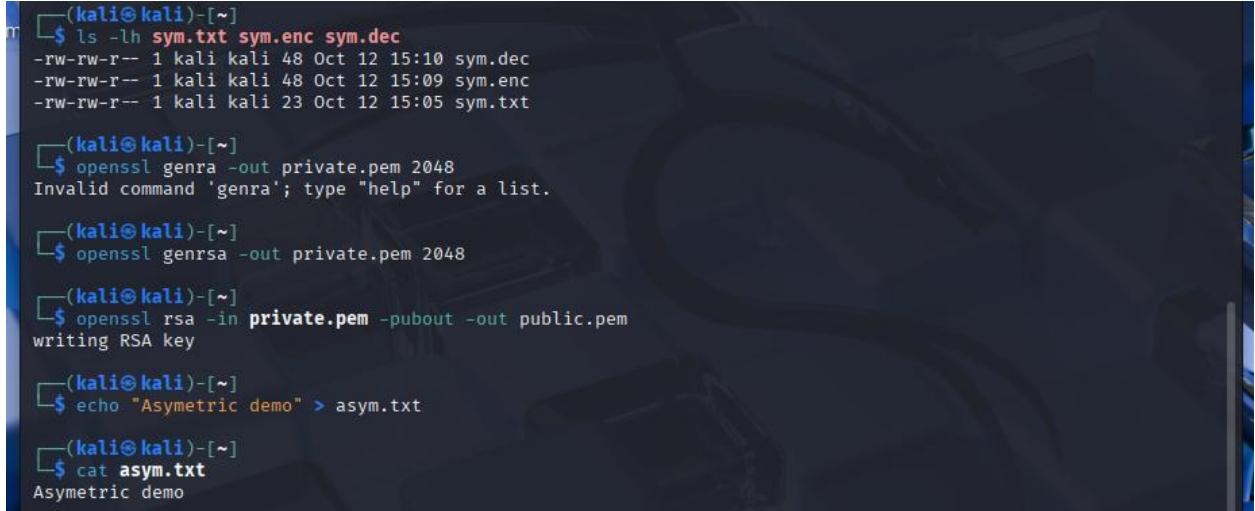
(kali㉿kali)-[~]
$ cat sym.dec
0^♦Zed_*****♦
♦,5♦♦&SM♦@♦l♦fl^*****g♦

(kali㉿kali)-[~]
$ ls -lh sym.txt sym.enc sym.dec
-rw-rw-r-- 1 kali kali 48 Oct 12 15:10 sym.dec
-rw-rw-r-- 1 kali kali 48 Oct 12 15:09 sym.enc
-rw-rw-r-- 1 kali kali 23 Oct 12 15:05 sym.txt

(kali㉿kali)-[~]
$ ss|
```

Screenshot showing successful decryption of “sym.enc” and restored plaintext output “Secret symmetric demo”.

### Step 3



```
(kali㉿kali)-[~]
└─$ ls -lh sym.txt sym.enc sym.dec
-rw-rw-r-- 1 kali kali 48 Oct 12 15:10 sym.dec
-rw-rw-r-- 1 kali kali 48 Oct 12 15:09 sym.enc
-rw-rw-r-- 1 kali kali 23 Oct 12 15:05 sym.txt

(kali㉿kali)-[~]
└─$ openssl genra -out private.pem 2048
Invalid command 'genra'; type "help" for a list.

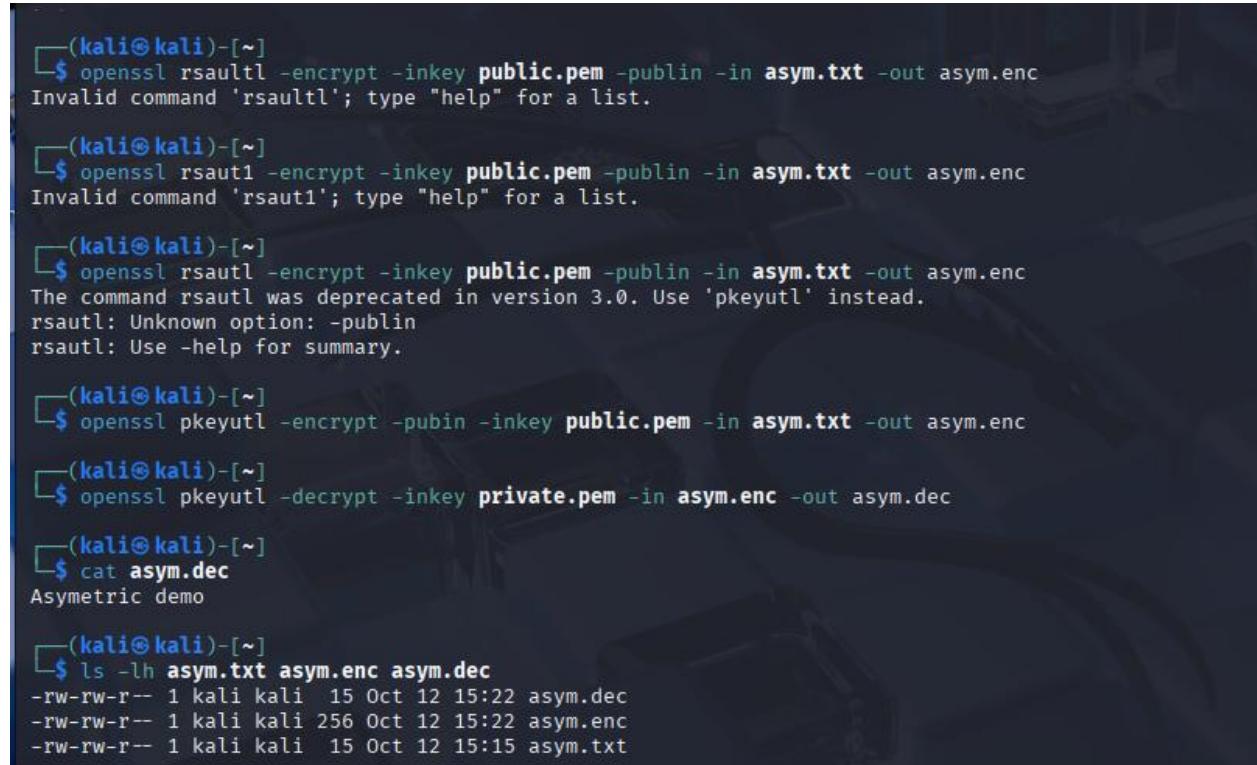
(kali㉿kali)-[~]
└─$ openssl genrsa -out private.pem 2048

(kali㉿kali)-[~]
└─$ openssl rsa -in private.pem -pubout -out public.pem
writing RSA key

(kali㉿kali)-[~]
└─$ echo "Asymmetric demo" > asym.txt

(kali㉿kali)-[~]
└─$ cat asym.txt
Asymmetric demo
```

*Screenshot showing RSA key generation and encryption of “asym.txt” using the public key.*



```
(kali㉿kali)-[~]
└─$ openssl rsautl -encrypt -inkey public.pem -pubin -in asym.txt -out asym.enc
Invalid command 'rsautl'; type "help" for a list.

(kali㉿kali)-[~]
└─$ openssl rsaut1 -encrypt -inkey public.pem -pubin -in asym.txt -out asym.enc
Invalid command 'rsaut1'; type "help" for a list.

(kali㉿kali)-[~]
└─$ openssl rsautl -encrypt -inkey public.pem -pubin -in asym.txt -out asym.enc
The command rsautl was deprecated in version 3.0. Use 'pkeyutl' instead.
rsautl: Unknown option: -pubin
rsautl: Use -help for summary.

(kali㉿kali)-[~]
└─$ openssl pkeyutl -encrypt -pubin -inkey public.pem -in asym.txt -out asym.enc

(kali㉿kali)-[~]
└─$ openssl pkeyutl -decrypt -inkey private.pem -in asym.enc -out asym.dec

(kali㉿kali)-[~]
└─$ cat asym.dec
Asymmetric demo

(kali㉿kali)-[~]
└─$ ls -lh asym.txt asym.enc asym.dec
-rw-rw-r-- 1 kali kali 15 Oct 12 15:22 asym.dec
-rw-rw-r-- 1 kali kali 256 Oct 12 15:22 asym.enc
-rw-rw-r-- 1 kali kali 15 Oct 12 15:15 asym.txt
```

*Screenshot confirming use of pkeyutl to encrypt and decrypt with RSA keys in OpenSSL 3.x, restoring “Asymmetric demo”.*

## Step 4:

```
(kali㉿kali)-[~]
└─$ openssl rsautl -encrypt -inkey public.pem -pubin -in asym.txt -out asym.enc
Invalid command 'rsautl'; type "help" for a list.

(kali㉿kali)-[~]
└─$ openssl rsautl -encrypt -inkey public.pem -pubin -in asym.txt -out asym.enc
The command rsautl was deprecated in version 3.0. Use 'pkeyutl' instead.
rsautl: Unknown option: -pubin
rsautl: Use -help for summary.

(kali㉿kali)-[~]
└─$ openssl pkeyutl -encrypt -pubin -inkey public.pem -in asym.txt -out asym.enc

(kali㉿kali)-[~]
└─$ openssl pkeyutl -decrypt -inkey private.pem -in asym.enc -out asym.dec

(kali㉿kali)-[~]
└─$ cat asym.dec
Asymmetric demo

(kali㉿kali)-[~]
└─$ ls -lh asym.txt asym.enc asym.dec
-rw-rw-r-- 1 kali kali 15 Oct 12 15:22 asym.dec
-rw-rw-r-- 1 kali kali 256 Oct 12 15:22 asym.enc
-rw-rw-r-- 1 kali kali 15 Oct 12 15:15 asym.txt

(kali㉿kali)-[~]
└─$ time dd if=/dev/zero bs=1M count=50 | nc 192.168.110.129 12345
50+0 records in
50+0 records out
52428800 bytes (52 MB, 50 MiB) copied, 1.31333 s, 39.9 MB/s
```

Screenshot displaying Kali terminal output from `time dd ... | nc ...` showing successful 50 MB transfer and elapsed time used for throughput calculation.

```
sfadmin@metasploitable:~$ ip a
: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        inet6 ::1/128 scope host
            valid_lft forever preferred_lft forever
: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast qlen 1000
    link/ether 00:0c:29:9f:0b:d4 brd ff:ff:ff:ff:ff:ff
    inet 192.168.110.129/24 brd 192.168.110.255 scope global eth0
        inet6 fe80::20c:29ff:fe9f:bd4/64 scope link
            valid_lft forever preferred_lft forever
sfadmin@metasploitable:~$ nc -l -p 12345 > /tmp/received.bin
```

## Step 5

```
(kali㉿kali)-[~]
└─$ ping -c 10 192.168.110.129
PING 192.168.110.129 (192.168.110.129) 56(84) bytes of data.
64 bytes from 192.168.110.129: icmp_seq=1 ttl=64 time=0.351 ms
64 bytes from 192.168.110.129: icmp_seq=2 ttl=64 time=0.230 ms
64 bytes from 192.168.110.129: icmp_seq=3 ttl=64 time=0.299 ms
64 bytes from 192.168.110.129: icmp_seq=4 ttl=64 time=0.319 ms
64 bytes from 192.168.110.129: icmp_seq=5 ttl=64 time=0.266 ms
64 bytes from 192.168.110.129: icmp_seq=6 ttl=64 time=0.249 ms
64 bytes from 192.168.110.129: icmp_seq=7 ttl=64 time=0.273 ms
64 bytes from 192.168.110.129: icmp_seq=8 ttl=64 time=0.247 ms
64 bytes from 192.168.110.129: icmp_seq=9 ttl=64 time=0.236 ms
64 bytes from 192.168.110.129: icmp_seq=10 ttl=64 time=0.256 ms

--- 192.168.110.129 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9218ms
rtt min/avg/max/mdev = 0.230/0.272/0.351/0.037 ms

(kali㉿kali)-[~]
└─$ sudo tc qdisc add dev eth0 root netem delay 100ms
[sudo] password for kali:
Sorry, try again.
[sudo] password for kali:

(kali㉿kali)-[~]
└─$ ping -c 5 192.168.110.129
PING 192.168.110.129 (192.168.110.129) 56(84) bytes of data.
64 bytes from 192.168.110.129: icmp_seq=1 ttl=64 time=101 ms
64 bytes from 192.168.110.129: icmp_seq=2 ttl=64 time=101 ms
64 bytes from 192.168.110.129: icmp_seq=3 ttl=64 time=101 ms
^[[B^[[B^[[B64 bytes from 192.168.110.129: icmp_seq=4 ttl=64 time=101 ms
64 bytes from 192.168.110.129: icmp_seq=5 ttl=64 time=101 ms

--- 192.168.110.129 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4007ms
rtt min/avg/max/mdev = 100.597/100.715/100.908/0.107 ms

(kali㉿kali)-[~]
└─$ sudo tc qdisc change dev eth0 root netem delay 200ms

(kali㉿kali)-[~]
└─$ ping -c 5 192.168.110.129
PING 192.168.110.129 (192.168.110.129) 56(84) bytes of data.
64 bytes from 192.168.110.129: icmp_seq=1 ttl=64 time=201 ms
64 bytes from 192.168.110.129: icmp_seq=2 ttl=64 time=201 ms
64 bytes from 192.168.110.129: icmp_seq=3 ttl=64 time=201 ms
64 bytes from 192.168.110.129: icmp_seq=4 ttl=64 time=201 ms
64 bytes from 192.168.110.129: icmp_seq=5 ttl=64 time=201 ms

--- 192.168.110.129 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4004ms
rtt min/avg/max/mdev = 200.656/200.744/200.820/0.064 ms

(kali㉿kali)-[~]
└─$ sudo tc qdisc del dev eth0 root
```

Screenshot showing ping results for baseline, 100 ms, and 200 ms delay tests on Kali using tc qdisc, demonstrating increased round-trip times as delay rises.

## Analysis Section

This lab connected the concepts of **source coding, channel capacity, encryption, and network flow** by combining theory with real-world Linux exercises.

In the written portion, Huffman coding illustrated how assigning variable-length codes to frequent symbols minimizes redundancy and lowers the total number of bits needed to represent data. Comparing Huffman and fixed-length encoding for the word *bookkeeper* showed that compression can reduce total bits by one-third, confirming that information theory principles directly improve efficiency.

The channel-capacity problem demonstrated how noise limits reliable transmission. Substituting different error probabilities into Shannon's equation revealed that as the bit-error rate increases, the maximum achievable throughput decreases. This built the foundation for understanding why encoding, compression, and error control must work together in communication systems.

The hands-on section reinforced these ideas in practice. The **compression demo** proved that repetitive data compresses effectively, while random data does not, linking physical file size to theoretical redundancy. **AES** and **RSA** experiments introduced data security: AES showed fast symmetric encryption for bulk data, and RSA demonstrated secure key exchange through asymmetric keys. Together they represented the balance between speed and security in networked systems.

The **channel-capacity experiment** using `nc` and `dd` quantified throughput on a virtual LAN. Sending compressed versus uncompressed files confirmed that compression reduces the amount of data transmitted, increasing apparent speed. Finally, the **data-flow and delay tests** using `ping`, `tc qdisc`, and `iperf3` illustrated how latency and queuing degrade performance, validating queueing-theory predictions where increased delay lowers effective bandwidth.

Overall, the lab showed how **information theory, encryption, and network performance** are intertwined. Efficient encoding reduces bits, secure encryption protects them, and understanding channel limits ensures they move reliably across a network. By integrating both analytical and experimental tasks, the lab provided a complete view of how data integrity, efficiency, and reliability are maintained in modern communication systems.

## Appendix Section

During the **Compression Demo (Part 2A)**, the `gzip` command was used on a repetitive text file to demonstrate how compression removes redundancy. Screenshots show the creation of

`test.txt`, the use of `gzip`, and the resulting file size comparison (`test.txt` vs `test.txt.gz`). This confirmed that compression improves data efficiency for transmission and storage.

In the **Symmetric Encryption (AES) section (Part 2B)**, AES-256-CBC was applied to encrypt and decrypt a text file using the same password. Screenshots include the encryption prompt, decryption output, and file verification, proving that symmetric encryption relies on one shared key for both processes.

For **Asymmetric Encryption (RSA) (Part 2C)**, RSA keys were generated using OpenSSL. The public key encrypted the file, and the private key successfully decrypted it, restoring the original text. Screenshots display key generation, encryption, decryption, and output verification, demonstrating the principle of public/private key pairs.

In the **Channel Capacity test (Part 2D)**, `nc` and `dd` commands were used to send a 50 MB file from Kali to Metasploitable. The recorded throughput and `stat` verification showed that compressed files transfer faster because fewer bits are transmitted, illustrating the relationship between data size and channel efficiency.

Finally, in the **Data Flow section (Part 2E)**, `ping` and `tc qdisc` commands simulated 100 ms and 200 ms of delay, while `iperf3` measured throughput under each condition. Screenshots show how latency increases round-trip time and reduces throughput, confirming that delay and queueing directly affect network performance.

## Conclusion

This lab combined theoretical and practical applications of data communication concepts. By completing all parts, I demonstrated how compression (Huffman coding) minimizes redundancy, encryption (AES & RSA) secures information, and network measurements (Netcat, iPerf3, and ping) reveal how bandwidth and delay affect real performance.

The experiments confirmed that:

- Compression improves effective throughput.
- Channel capacity decreases as noise or delay increases.
- Symmetric encryption is faster for large data, while asymmetric encryption provides secure key exchange.
- Artificial latency directly increases round-trip time and lowers throughput.

Overall, the lab showed how the ideas of **source coding, channel capacity, and data flow** work together to define efficiency, reliability, and security in digital communication systems.