

# CSE 3501

## INFORMATION SECURITY ANALYSIS & AUDIT



### Midterm Lab

Group B

L9+L10 | PLBG04

FALL SEMESTER 2021-22

by

**SHARADINDU ADHIKARI**

19BCE2105

## Group B

## Question 1

1. **Aim:** To implement the RSA algorithm.
2. **Objective:** After some research and studying RSA briefly, I've decided to go forward with implementing it in C++.
3. **Procedure:**  
We know right, RSA Algorithm is an example for Public Key Encryption algorithm; so here we are supposed to find two keys:  
1) Public Key which is used at encryption, & 2) Private Key which is used at decryption.

Step 1: Select two large Primes P, Q  
Step 2: Calculate  $n = P * Q$  &  $O(n) = (P-1) * (Q-1)$   
Step 3: Assume e and d (Public and Private Key).  
Step 4: Encrypt the Plain Text using Public Key e.  
Step 5: Decrypt the Cipher Text using Private Key d.

## 4. Code:

```
#include<bits/stdc++.h>
using namespace std;
#define int unsigned long long

bool check_prime(int n){
    for(int i = 2; i <= n/2; i++){
        if(n % i == 0){
            return false;
        }
    }
    return true;
}

int32_t main(void){
    srand(time(0));
    int p = 4;
    while(p == 1 || p == 0 || !check_prime(p)){
        p = rand() % 15;
    }

    cout << p << endl;
    int q = 4;
    while(q == 0 || q == 1 || q == p || !check_prime(q)){
        q = rand() % 15;
    }

    cout << q << endl;
    int n = p * q;
    int phi_n = (p-1) * (q-1);
    int e = 2;
```

```
while(e < phi_n){
    if(__gcd(e, phi_n) == 1) break;
    e++;
}

int mssg;
cout << "Please enter the message: " << endl;
cin >> mssg;
cout << endl << mssg << " - Original Message" << endl;
int d = 1;

while(true){
    int temp = (d*e) % phi_n;
    if(temp == 1) break;
    d++;
}

//cout << "Public Key " << e << " " << n << endl;
//cout << "Private Key " << d << endl;
//cout << "Phi " << phi_n << endl;
//cout << (d*e) % phi_n << endl;

int c = (unsigned long)pow(mssg, e);
//cout << c << endl;
c = c % n;
cout << c << " - Encrypted Data" << endl;

int plain = (unsigned long)pow(c, d);
plain = plain % n;
//cout << plain << endl;
cout << plain << " - Message after Decrypting" << endl;
return 0;
}
```

```
File Edit Selection View Go Run Terminal Help
algo.cpp - Visual Studio Code

lab2.java alight1.html G+ algo.cpp 2 X
C:\Users\shara> OneDrive\ Desktop\ G+ algo.cpp > main(void)
1 #include<bits/stdc++.h>
2
3 using namespace std;
4
5 #define int unsigned long long
6
7 bool check_prime(int n){
8
9     for(int i = 2; i <= n/2; i++){
10         if(n % i == 0){
11             return false;
12         }
13     }
14     return true;
15 }
16
17
18 int32_t main(void){
19     srand(time(0));
20     int p = 4;
21     while(p == 1 || p == 0 || !check_prime(p)){
22         p = rand() % 15;
23     }
24
25     cout << p << endl;
26
27     int q = 4;
28     while(q == 0 || q == 1 || q == p || !check_prime(q)){
29         q = rand() % 15;
30     }
31
32     cout << q << endl;
33
34     int n = p * q;
35
36     int phi_n = (p-1) * (q-1);
37     int e = 2;
```

```

37 int phi_n = (p-1) * (q-1);
38 int e = 2;
39 while(e < phi_n){
40     if(__gcd(e, phi_n) == 1) break;
41     e++;
42 }
43
44 int mssg;
45 cout << "Please enter the message: " << endl;
46 cin >> mssg;
47 cout << endl << mssg << " - Original Message" << endl;
48 int d = 1;
49
50 while(true){
51     int temp = (d*e) % phi_n;
52     if(temp == 1) break;
53     d++;
54 }
55
56 //cout << "Public Key " << e << " " << n << endl;
57 //cout << "Private Key " << d << endl;
58 //cout << "Phi " << phi_n << endl;
59 //cout << (d*e) % phi_n << endl;
60 int c = (unsigned long)pow(mssg, e);
61 //cout << c << endl;
62 c = c % n;
63 cout << c << " - Encrypted Data" << endl;
64
65 int plain = (unsigned long)pow(c, d);
66 plain = plain % n;
67 //cout << plain << endl;
68 cout << plain << " - Message after Decryption" << endl;
69
70
71
72 return 0;
73

```

## 5. Results screenshot:

```

Windows PowerShell
Copyright (c) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\shara> cd onedrive
PS C:\Users\shara\onedrive> cd desktop
PS C:\Users\shara\onedrive\desktop> g++ -o algo algo.cpp
PS C:\Users\shara\onedrive\desktop> ./algo
2
7
Please enter the message:
6
6 - Original Message
6 - Encrypted Data
6 - Message after Decryption
PS C:\Users\shara\onedrive\desktop>

```

6. **Conclusion:** While establishing RSA key pairs, usage keys and general-purpose keys are generally integrated. In usage RSA keys, two key pairs are used for encryption and signatures. I've tried to demonstrate the same using C++. Hope you liked it.

## Question 2

1. **Aim:** To all UDP related captures using packet Sniffer in Wireshark.
2. **Objective:** To demonstrate Capture and Display filters in Wireshark, while being connected to WiFi.
3. **Procedure:**
  1. Start up the Wireshark program (select an interface and press start to capture packets).
  2. Start up our favourite browser (Edge)
  3. In the browser, go to any homepage
  4. After browser has displayed the webpage, stop Wireshark packet capture by selecting stop in the Wireshark capture window. This will cause the Wireshark capture window to disappear and the main Wireshark window to display all packets capture.
  5. Apply all the filters.
  6. Take Implementation screenshots of the all the stuff asked in the question.

## 4. Implementation Screenshots:

### Capture Filter:

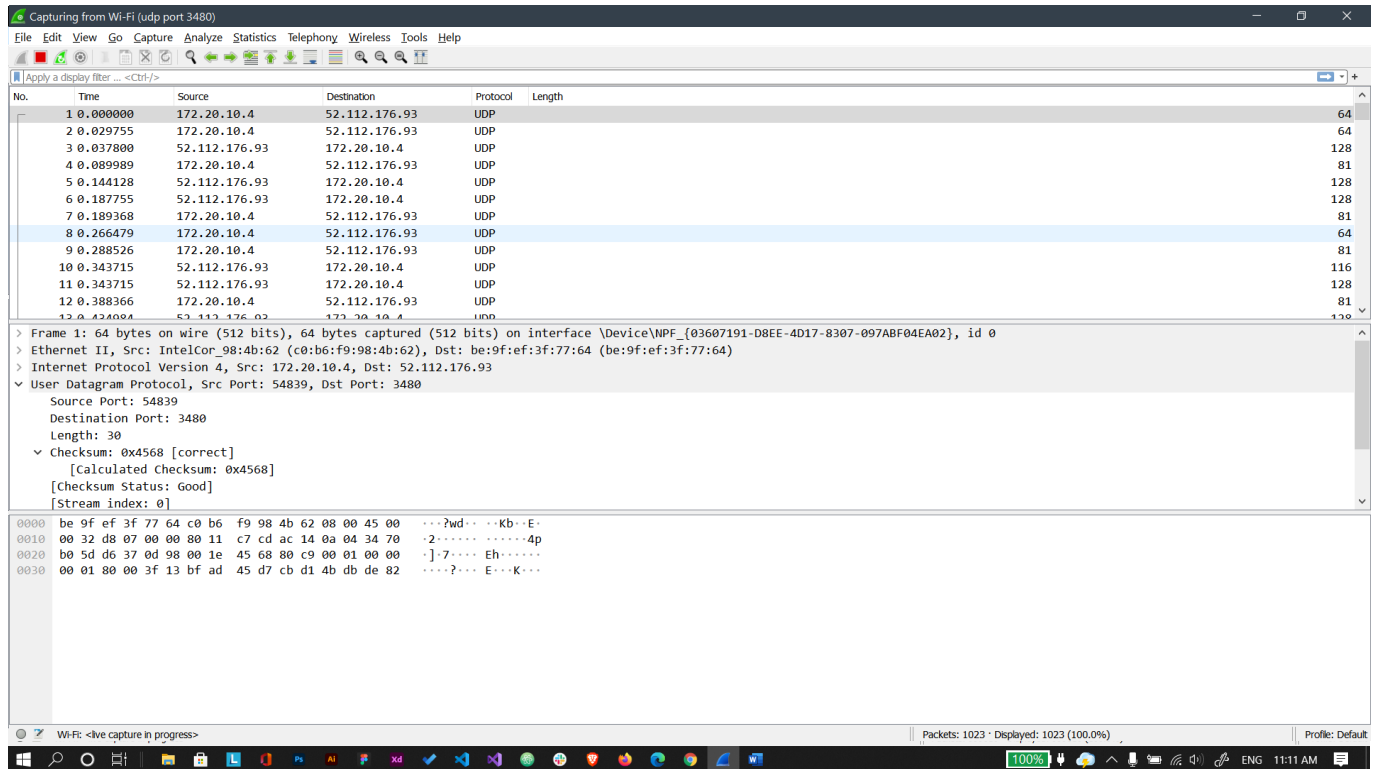
### UDP:

The screenshot shows the Wireshark interface with a packet capture filter applied: `udp`. The packet list shows several UDP packets. The selected packet (No. 128) is a User Datagram Protocol packet from Source Port 3480 to Destination Port 54839. The packet details pane shows the User Datagram Protocol section with the following information:

- Source Port: 3480
- Destination Port: 54839
- Length: 94
- Checksum: 0xebd5 [correct]
- [calculated Checksum: 0xebd5]
- [Checksum Status: Good]
- [Stream index: 0]

The packet bytes pane shows the raw data of the packet, including the Ethernet II header, Internet Protocol Version 4 header, and the User Datagram Protocol payload.

## UDP Port 3480:



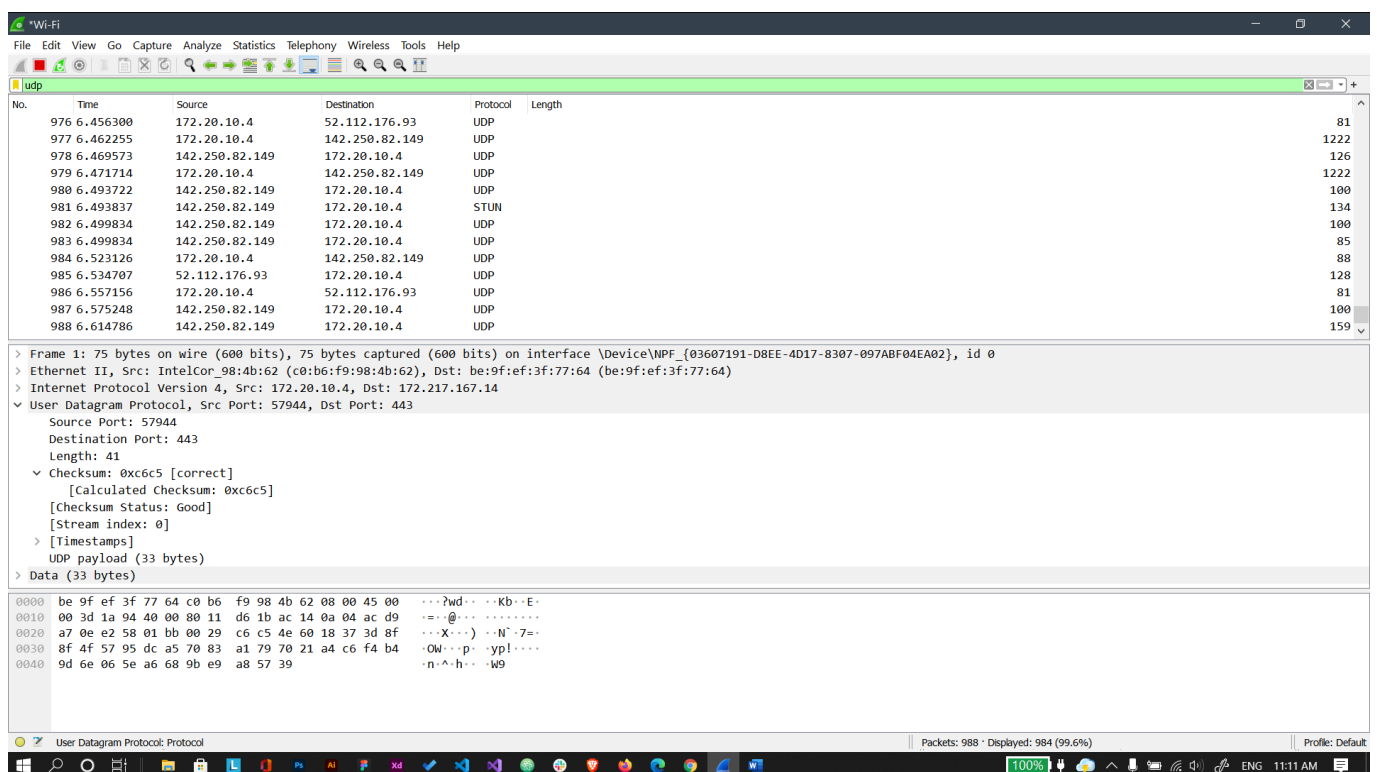
Wireshark capture of UDP traffic on port 3480. The capture is filtered by 'udp port 3480'. The packet list shows 17 packets, with packet 8 selected. The packet details pane shows the structure of the selected packet: Ethernet II, Internet Protocol Version 4, and User Datagram Protocol (Src Port: 54839, Dst Port: 3480). The packet bytes pane shows the raw data in hexadecimal and ASCII.

No.	Time	Source	Destination	Protocol	Length
1	0.000000	172.20.10.4	52.112.176.93	UDP	64
2	0.029755	172.20.10.4	52.112.176.93	UDP	64
3	0.037800	52.112.176.93	172.20.10.4	UDP	128
4	0.089989	172.20.10.4	52.112.176.93	UDP	81
5	0.144128	52.112.176.93	172.20.10.4	UDP	128
6	0.187755	52.112.176.93	172.20.10.4	UDP	128
7	0.189368	172.20.10.4	52.112.176.93	UDP	81
8	0.266479	172.20.10.4	52.112.176.93	UDP	64
9	0.288526	172.20.10.4	52.112.176.93	UDP	81
10	0.343715	52.112.176.93	172.20.10.4	UDP	116
11	0.343715	52.112.176.93	172.20.10.4	UDP	128
12	0.388366	172.20.10.4	52.112.176.93	UDP	81
13	0.426084	52.112.176.93	172.20.10.4	UDP	128

Frame 8: 64 bytes on wire (512 bits), 64 bytes captured (512 bits) on interface \Device\NPF\_{03607191-D8EE-4D17-8307-097ABF04EA02}, id 0  
> Ethernet II, Src: IntelCor\_98:4b:62 (c0:b6:f9:98:4b:62), Dst: be:9f:ef:3f:77:64 (be:9f:ef:3f:77:64)  
> Internet Protocol Version 4, Src: 172.20.10.4, Dst: 52.112.176.93  
User Datagram Protocol, Src Port: 54839, Dst Port: 3480  
Source Port: 54839  
Destination Port: 3480  
Length: 30  
Checksum: 0x4568 [correct]  
[Calculated Checksum: 0x4568]  
[Checksum Status: Good]  
[Stream index: 0]  
0000 be 9f ef 3f 77 64 c0 b6 f9 98 4b 62 08 00 45 00 ...?wd...Kb...E-  
0010 00 32 d8 07 00 00 00 11 c7 cd ac 14 0a 04 34 70 -2.....4p  
0020 b0 5d d6 37 0d 98 00 1e 45 68 80 c9 00 01 00 00 -]7....Eh.....  
0030 00 01 80 00 3f 13 bf ad 45 d7 cb d1 4b db de 82 ....?...E...K...

## Display filters:

## UDP:

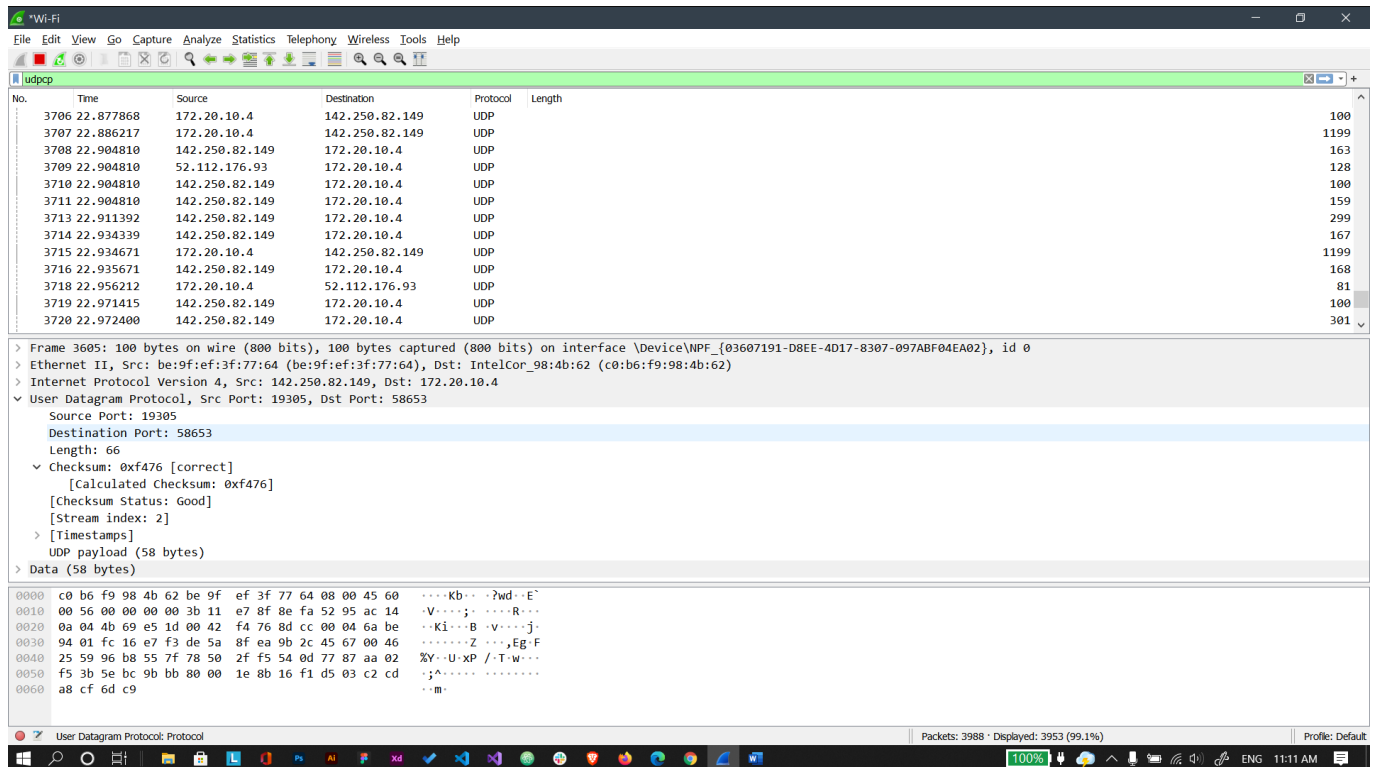


Wireshark capture of UDP traffic. The capture is filtered by 'udp'. The packet list shows 17 packets, with packet 1 selected. The packet details pane shows the structure of the selected packet: Ethernet II, Internet Protocol Version 4, and User Datagram Protocol (Src Port: 57944, Dst Port: 443). The packet bytes pane shows the raw data in hexadecimal and ASCII.

No.	Time	Source	Destination	Protocol	Length
976	6.456300	172.20.10.4	52.112.176.93	UDP	81
977	6.462255	172.20.10.4	142.250.82.149	UDP	1222
978	6.469573	142.250.82.149	172.20.10.4	UDP	126
979	6.471714	172.20.10.4	142.250.82.149	UDP	1222
980	6.493722	142.250.82.149	172.20.10.4	UDP	100
981	6.493837	142.250.82.149	172.20.10.4	STUN	134
982	6.499834	142.250.82.149	172.20.10.4	UDP	100
983	6.499834	142.250.82.149	172.20.10.4	UDP	85
984	6.523126	172.20.10.4	142.250.82.149	UDP	88
985	6.534707	52.112.176.93	172.20.10.4	UDP	128
986	6.557156	172.20.10.4	52.112.176.93	UDP	81
987	6.575248	142.250.82.149	172.20.10.4	UDP	100
988	6.614786	142.250.82.149	172.20.10.4	UDP	159

Frame 1: 75 bytes on wire (600 bits), 75 bytes captured (600 bits) on interface \Device\NPF\_{03607191-D8EE-4D17-8307-097ABF04EA02}, id 0  
> Ethernet II, Src: IntelCor\_98:4b:62 (c0:b6:f9:98:4b:62), Dst: be:9f:ef:3f:77:64 (be:9f:ef:3f:77:64)  
> Internet Protocol Version 4, Src: 172.20.10.4, Dst: 172.217.167.14  
User Datagram Protocol, Src Port: 57944, Dst Port: 443  
Source Port: 57944  
Destination Port: 443  
Length: 41  
Checksum: 0xc6c5 [correct]  
[Calculated Checksum: 0xc6c5]  
[Checksum Status: Good]  
[Stream index: 0]  
[Timestamps]  
UDP payload (33 bytes)  
> Data (33 bytes)  
0000 be 9f ef 3f 77 64 c0 b6 f9 98 4b 62 08 00 45 00 ...?wd...Kb...E-  
0010 00 3d 1a 94 40 00 80 11 d6 1b ac 14 0a 04 ac d9 -...@.....  
0020 a7 0e e2 58 01 bb 00 29 c6 c5 4e 60 18 37 3d 8f ...X...N~7=  
0030 8f 4f 57 95 dc a5 70 83 a1 79 70 21 a4 c6 f4 b4 -OW...p...yp!....  
0040 9d 6e 06 5e a6 68 9b e9 a8 57 39 -n^h...w9

## UDPCP:



Wireshark capture of UDPCP traffic. The packet list shows a series of UDP packets from 172.20.10.4 to 142.250.82.149. The selected packet (No. 3706) is a User Datagram Protocol (UDP) packet with source port 19305 and destination port 58653. The packet details show the User Datagram Protocol (UDP) header and the Data payload (58 bytes).

No.	Time	Source	Destination	Protocol	Length
3706	22.877868	172.20.10.4	142.250.82.149	UDP	100
3707	22.886217	172.20.10.4	142.250.82.149	UDP	1199
3708	22.904810	142.250.82.149	172.20.10.4	UDP	163
3709	22.904810	52.112.176.93	172.20.10.4	UDP	128
3710	22.904810	142.250.82.149	172.20.10.4	UDP	100
3711	22.904810	142.250.82.149	172.20.10.4	UDP	159
3713	22.911392	142.250.82.149	172.20.10.4	UDP	299
3714	22.934339	142.250.82.149	172.20.10.4	UDP	167
3715	22.934671	172.20.10.4	142.250.82.149	UDP	1199
3716	22.935671	142.250.82.149	172.20.10.4	UDP	168
3718	22.956212	172.20.10.4	52.112.176.93	UDP	81
3719	22.971415	142.250.82.149	172.20.10.4	UDP	100
3720	22.972400	142.250.82.149	172.20.10.4	UDP	301

Frame 3605: 100 bytes on wire (800 bits), 100 bytes captured (800 bits) on interface \Device\NPF\_{03607191-D8EE-4D17-8307-097ABF04EA02}, id 0

Ethernet II, Src: be:9f:ef:3f:77:64 (be:9f:ef:3f:77:64), Dst: IntelCor\_98:4b:62 (c0:b6:f9:98:4b:62)

Internet Protocol Version 4, Src: 142.250.82.149, Dst: 172.20.10.4

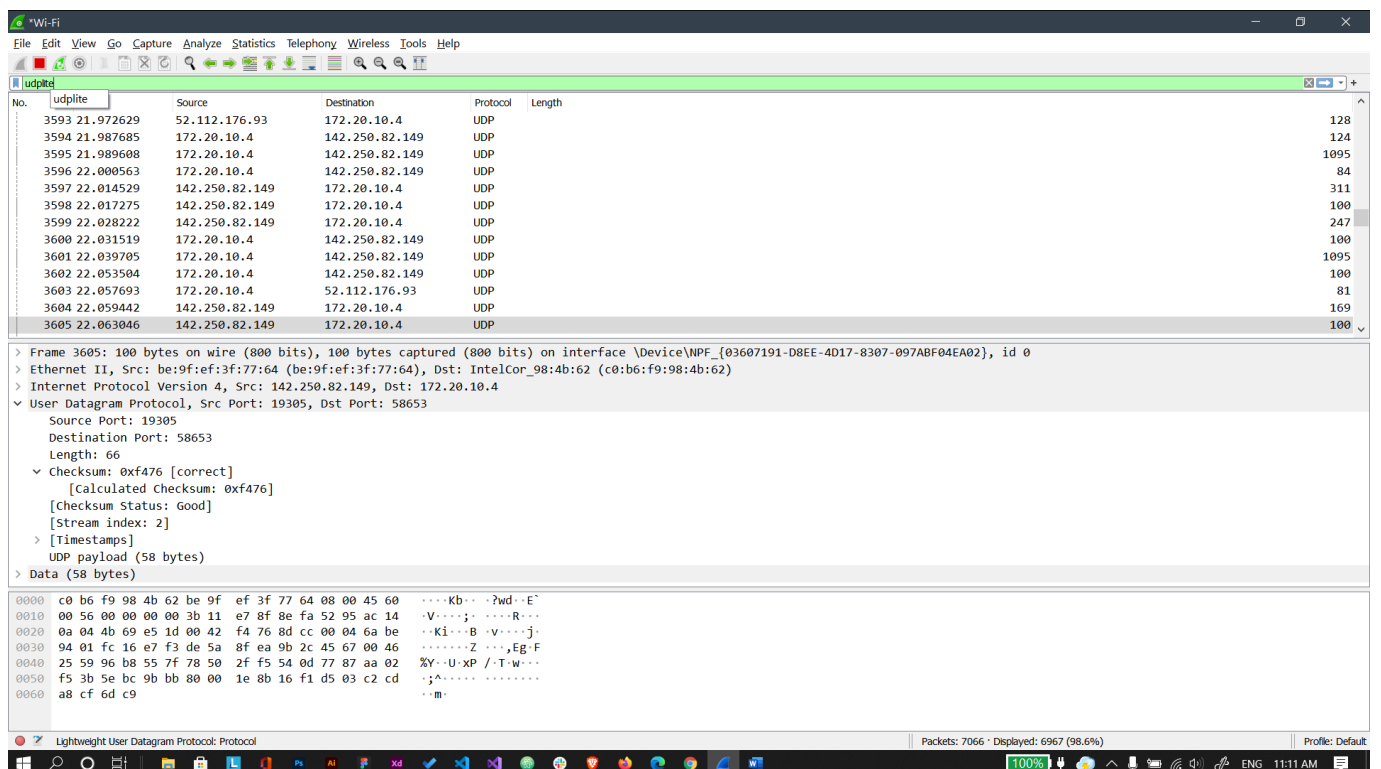
User Datagram Protocol, Src Port: 19305, Dst Port: 58653

Source Port: 19305  
Destination Port: 58653  
Length: 66  
Checksum: 0xf476 [correct]  
[Calculated Checksum: 0xf476]  
[Checksum Status: Good]  
[Stream index: 2]  
[Timestamps]  
UDP payload (58 bytes)

Data (58 bytes)

```
0000 c0 b6 f9 98 4b 62 be 9f ef 3f 77 64 08 00 45 60  ....Kb...?wd..E`
0010 00 56 00 00 00 00 3b 11 e7 8f 8e fa 52 95 ac 14  ..V....;....R...
0020 0a 04 4b 69 e5 1d 00 42 f4 76 8d cc 00 04 6a be  ..K1...B..v....j-
0030 94 01 fc 16 e7 f3 de 5a 8f ea 9b 2c 45 67 00 46  ....Z....Eg.F
0040 25 59 96 b8 55 7f 78 50 2f f5 54 0d 77 87 aa 02  %Y..U-xP /-T.w...
0050 f5 3b 5e bc 9b bb 80 00 1e 8b 16 f1 d5 03 c2 cd  ;^.....
0060 a8 cf 6d c9  ..m.
```

## UDPLite:



Wireshark capture of UDPLite traffic. The packet list shows a series of UDP packets from 52.112.176.93 to 172.20.10.4. The selected packet (No. 3593) is a User Datagram Protocol (UDP) packet with source port 19305 and destination port 58653. The packet details show the User Datagram Protocol (UDP) header and the Data payload (58 bytes).

No.	Time	Source	Destination	Protocol	Length
3593	21.972629	52.112.176.93	172.20.10.4	UDP	128
3594	21.987685	172.20.10.4	142.250.82.149	UDP	124
3595	21.989608	172.20.10.4	142.250.82.149	UDP	1095
3596	22.000563	172.20.10.4	142.250.82.149	UDP	84
3597	22.014529	142.250.82.149	172.20.10.4	UDP	311
3598	22.017275	142.250.82.149	172.20.10.4	UDP	100
3599	22.028222	142.250.82.149	172.20.10.4	UDP	247
3600	22.031519	172.20.10.4	142.250.82.149	UDP	100
3601	22.039705	172.20.10.4	142.250.82.149	UDP	1095
3602	22.053504	172.20.10.4	142.250.82.149	UDP	100
3603	22.057693	172.20.10.4	52.112.176.93	UDP	81
3604	22.059442	142.250.82.149	172.20.10.4	UDP	169
3605	22.063046	142.250.82.149	172.20.10.4	UDP	100

Frame 3605: 100 bytes on wire (800 bits), 100 bytes captured (800 bits) on interface \Device\NPF\_{03607191-D8EE-4D17-8307-097ABF04EA02}, id 0

Ethernet II, Src: be:9f:ef:3f:77:64 (be:9f:ef:3f:77:64), Dst: IntelCor\_98:4b:62 (c0:b6:f9:98:4b:62)

Internet Protocol Version 4, Src: 142.250.82.149, Dst: 172.20.10.4

User Datagram Protocol, Src Port: 19305, Dst Port: 58653

Source Port: 19305  
Destination Port: 58653  
Length: 66  
Checksum: 0xf476 [correct]  
[Calculated Checksum: 0xf476]  
[Checksum Status: Good]  
[Stream index: 2]  
[Timestamps]  
UDP payload (58 bytes)

Data (58 bytes)

```
0000 c0 b6 f9 98 4b 62 be 9f ef 3f 77 64 08 00 45 60  ....Kb...?wd..E`
0010 00 56 00 00 00 00 3b 11 e7 8f 8e fa 52 95 ac 14  ..V....;....R...
0020 0a 04 4b 69 e5 1d 00 42 f4 76 8d cc 00 04 6a be  ..K1...B..v....j-
0030 94 01 fc 16 e7 f3 de 5a 8f ea 9b 2c 45 67 00 46  ....Z....Eg.F
0040 25 59 96 b8 55 7f 78 50 2f f5 54 0d 77 87 aa 02  %Y..U-xP /-T.w...
0050 f5 3b 5e bc 9b bb 80 00 1e 8b 16 f1 d5 03 c2 cd  ;^.....
0060 a8 cf 6d c9  ..m.
```

## UDPencap:

Wireshark capture of UDP packets. The packet list shows several UDP packets from 172.20.10.4 to 142.250.82.149. Packet 3605 is selected, showing details for Ethernet II, Internet Protocol Version 4, User Datagram Protocol, and Data (58 bytes). The packet bytes pane shows the raw data in hexadecimal and ASCII.

No.	udpencap	Source	Destination	Protocol	Length
3593	21.972629	52.112.176.93	172.20.10.4	UDP	128
3594	21.987685	172.20.10.4	142.250.82.149	UDP	124
3595	21.989608	172.20.10.4	142.250.82.149	UDP	1095
3596	22.000563	172.20.10.4	142.250.82.149	UDP	84
3597	22.014529	142.250.82.149	172.20.10.4	UDP	311
3598	22.017275	142.250.82.149	172.20.10.4	UDP	100
3599	22.028222	142.250.82.149	172.20.10.4	UDP	247
3600	22.031519	172.20.10.4	142.250.82.149	UDP	100
3601	22.039705	172.20.10.4	142.250.82.149	UDP	1095
3602	22.053504	172.20.10.4	142.250.82.149	UDP	100
3603	22.057693	172.20.10.4	52.112.176.93	UDP	81
3604	22.059442	142.250.82.149	172.20.10.4	UDP	169
3605	22.063046	142.250.82.149	172.20.10.4	UDP	100

Frame 3605: 100 bytes on wire (800 bits), 100 bytes captured (800 bits) on interface \Device\NPF\_{03607191-D8EE-4D17-8307-097ABF04EA02}, id 0  
 Ethernet II, Src: be:9f:ef:3f:77:64 (be:9f:ef:3f:77:64), Dst: IntelCor\_98:4b:62 (c0:b6:f9:98:4b:62)  
 Internet Protocol Version 4, Src: 142.250.82.149, Dst: 172.20.10.4  
 User Datagram Protocol, Src Port: 19305, Dst Port: 58653  
 Source Port: 19305  
 Destination Port: 58653  
 Length: 66  
 Checksum: 0xf476 [correct]  
 [Calculated Checksum: 0xf476]  
 [Checksum Status: Good]  
 [Stream index: 2]  
 [Timestamps]  
 UDP payload (58 bytes)  
 Data (58 bytes)

0000 c0 b6 f9 98 4b 62 be 9f ef 3f 77 64 08 00 45 60 .....Kb...?wd..E  
 0010 00 56 00 00 00 00 3b 11 e7 8f 8e fa 52 95 ac 14 ..V....j...R..  
 0020 0a 04 4b 69 e5 1d 00 42 f4 76 8d cc 00 04 6a be ..Ki...B..v....j.  
 0030 94 01 fc 16 e7 f3 de 5a 8f ea 9b 2c 45 67 00 46 .....Z...Eg.F  
 0040 25 59 96 b8 55 7f 78 50 2f f5 54 0d 77 87 aa 02 %Y..U..XP /-T-w..  
 0050 f5 3b 5e bc 9b bb 80 00 1e 8b 16 f1 d5 03 c2 cd ;^.....  
 0060 a8 cf 6d c9 .....m.

5. **Conclusion:** As I've demonstrated here using Wireshark, UDP packets can be filtered greatly using the various filters available.