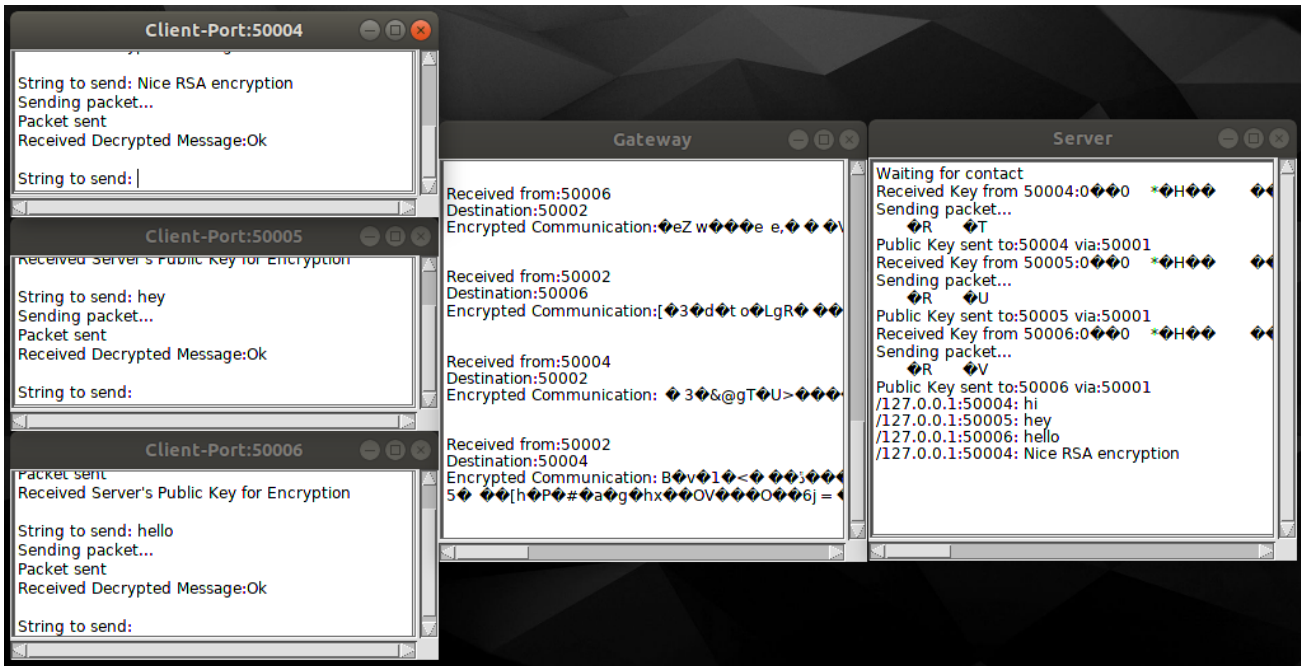
CS2031 - Telecommunications II - Assignment #1

Conor Gildea - gildeaco
gildeaco@tcd.ie - github.com/csigildea

November 13, 2017

1 Assignment #1 - Finished Product and Milestones



```
String to send: Nice RSA encryption
Sending packet...
Packet sent
Received Decrypted Message:Ok

String to send: |
```

```
Received Server's Public Key for Encryption

String to send: hey
Sending packet...
Packet sent
Received Decrypted Message:Ok

String to send:
```

```
Packet sent
Received Server's Public Key for Encryption

String to send: hello
Sending packet...
Packet sent
Received Decrypted Message:Ok

String to send:
```

```
Received from:50006
Destination:50002
Encrypted Communication:�eZ w����e e,� � �\

Received from:50002
Destination:50006
Encrypted Communication:[�3�d�t o�LgR� ��

Received from:50004
Destination:50002
Encrypted Communication:  � 3�&@gT�U>���

Received from:50002
Destination:50004
Encrypted Communication: B�v�1�<� ��ȝ��
5�  ��[h�P�#�a�g�hx��OV���O���6j =
```

```
Waiting for contact
Received Key from 50004:0��0    *�H��      ��
Sending packet...
     �R     �T
Public Key sent to:50004 via:50001
Received Key from 50005:0��0    *�H��      ��
Sending packet...
     �R     �U
Public Key sent to:50005 via:50001
Received Key from 50006:0��0    *�H��      ��
Sending packet...
     �R     �V
Public Key sent to:50006 via:50001
/127.0.0.1:50004: hi
/127.0.0.1:50005: hey
/127.0.0.1:50006: hello
/127.0.0.1:50004: Nice RSA encryption
```

Figure 1: Multiple Clients communicating via RSA encrypted messages via the Gateway.

Milestones I achieved in this Assignment:

- Creating a packet from client to server
- Encapsulating the packet to the server, to go to the gateway from the client.
- Decapsulating/Unpacking the packet, at the gateway for the client/server.
- Communicating between client to the gateway and to the server and vice versa.
- Understanding of sockets,datagram packets and threads
- Stop and Wait between client and server

Extra Milestones I achieved in this Assignment:

- **Multiple Clients**
- **Implementation of TCP - "Styled" Headers**
- **Checksum Implementation**
- **Unexpected Sequence Number Implementation**
- **Public and Private RSA key generation**
- **Two-Way RSA Encrypted Communication between Client and Server**

2 Client, Gateway and Server Code Process Explained

As these processes are threaded, trying to explain my code in psuedocode would result in pseudocode extremely similar to the code I produced, I believe it could be quite confusing to do this. Due to this, I am detailing the explanation of my code in psuedocode in bulletpoint/paragraph form, in the order it is run. I believe this should be much easier to follow, in this case.

2.1 Client

- Finds Unused Port
- Designates Port Number to itself
- Starts Terminal with Port Number
- Generates a Keypair
- Sends Public Key from Keypair to the Server (packeting explained further below)
- Stop and Wait Protocol activated - Keeps sending until acknowledgement packet and acknowledgement number received from server.
- In the acknowledgement packet from the server, the server's public key should be present.
- Server's public key is then saved for future encryption of messages, if this occurs correctly receivedServerPublicKey is set to true.
- Client terminal then waits for user to enter a message
- Client takes the user's enter message encrypts it using the server's public key, this is considered the payload.
- A header is then generatated, source port being the client's own port, destination port being the server's port. Sequence Number, Acknowledgement Number and Checksum are also generated and added.
- The header and the payload are then combined to form the packet
- Packet Encapsulation now occurs where another header is added to the packet, this time with the destination port being set to the gateway port. This is necessary to transport the packet correctly
- Encapsulated Packet is then sent on to the gateway and the Stop and Wait Protocol is activated again
- If no response is received within the given timeframe (3 secs in this case) the client keeps sending and leaves a message on the terminal mentioning the resending of the packet.
- Once a message is received from the server via the gateway this message is then decrypted using the client's private key corresponding to the public key it sent to the server. This decrypted message is then printed to the terminal.
- The client then gets more user input and repeats the process.

Client - Edge Cases/Error Handling/Additional Notes:

- No acknowledgement Packet Received - Stop and Wait contiunues sending packet if none are received within the timeframe
- Dynamicly finds a port - The port isn't predefined in this program meaning the client will always find a port free so you can use as many clients as you can find within the range of free ports.
- An Acknowledgement number from the server can request a packet be resent if the sequence number or the checksum value in the header is incorrect/unexpected, the client facilitates this resending, until the server sends a continue communicating acknowledgement request back.
- Only works with a valid RSA Public Key, that the server should send.
- Handles sequence numbers and checksum numbers being compared against what they should be.
- Use of RSA encryption prevents people snooping on the messages being sent between the client and the server.

2.2 Gateway

My implementation of my gateway was previously much more complex however the addition of proper use of headers, it reduced the amount of code required substantially whilst keeping the exact same functionality

- The gateway receives a packet from the client/server
- The gateway then decapsulates this packet to remove the now uncessary header that was used to transport it to the gateway.
- The gateway prints the port number it is being sent from and the destination it is going to, in the terminal.
- The content within the packet is printed to the terminal.
- *This content is normally meaningless to read, as it either a public key being exchanged between the client/server or it is an RSA encrypted message between the two.*
- The packet is then forwarded onwards to the destination listed in the decapsulated packet header.
- The gateway waits until another packet is sent for it to transport it correctly.

2.3 Server

The implementation of the server is very similiar to the client however it is also catering for storing multiple ports with corresponding sequence numbers and public keys, in order to communicate in the correct order and to encrypt messages to be delievered to users properly.

Due to this some of the most notable additions to the server are the arraylists of the ports,sequenceNumbers and the clients' public keys.

- Server starts and generates a key pair.
- When the server recieves a public key from a client, it saves the clients port, sequence number and public key in corresponding indexing arraylists for easy reference.
- Server then sends the client back and acknowledgement with the server's public key.
- Server waits for additional messages from a client.
- Any content in a packet from a client who has already communicated with the server are now considered to be encrypted using the server's public key.
- The server checks is the checksum and the sequence number of this packet correct, if not it requests the correct packet to be resent and waits for the expected sequence number/correct checksum value.
- The server decrypts the content of the packet it received, using it's own private key.
- The server forms a response message (always "Ok") and RSA encrypts this message using the client's public key.
- This payload is then added to a corresponding header with the client's port as the destination and the server's port as the source port. With the corresponding correct sequence number, acknowledgement number and checksum.
- This packet is then encapsulated with the gateway port being set as the destination port in the encapsulating header.
- The server then waits for additional packets to be received from clients and repeats the process detailed above in order to handle them correctly.

3 Main Factors of this Assignment

3.1 Header Implementation

Originally I was attaching my additional information that would have gone into a header on to the end of my string in my packet. I found this method just really messy and added a unnecessary level of complexitiy. Despite the previous method working, it wasn't as easily scablable and required much more code for the same functionality. I decided to rework my entire project to implement headers and I am very happy I did.

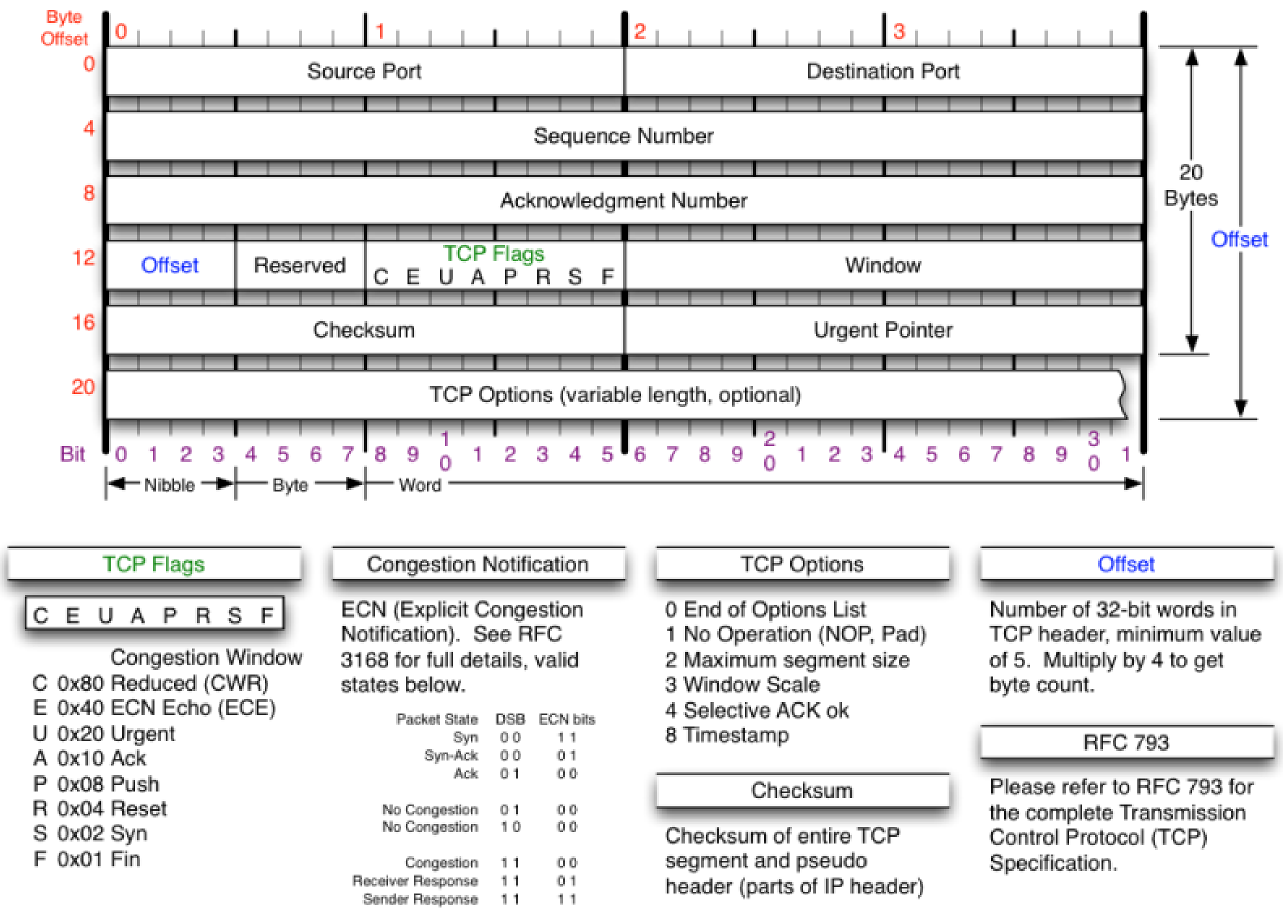
I generated my own protocol for this assignment based on TCP - Transmission Control Protocol. I was considering using UDP however I wanted to try and include an acknowledgement number inside my header.

For ease of implementation I included a Header class in this project, this merely consisted of a byte array inside a Header object however it allowed me to access a range of different functions to set/get any data from my header via the client, gateway or server.

This data in my header included:

- **Bytes 0 - 1:** Source Port
- **Bytes 2 - 3:** Destination Port
- **Bytes 4 - 7:** Sequence Number
- **Bytes 8 - 11:** Acknowledgement Number
- **Bytes 12 - 13:** Checksum

As we can see my implementation of an edited Transmission Control Protocol is very similiar, in comparision, to the actual TCP.



TCP Flags

C E U A P R S F

Congestion Window
C 0x80 Reduced (CWR)
E 0x40 ECN Echo (ECE)
U 0x20 Urgent
A 0x10 Ack
P 0x08 Push
R 0x04 Reset
S 0x02 Syn
F 0x01 Fin

Congestion Notification

ECN (Explicit Congestion Notification). See RFC 3168 for full details, valid states below.

Packet State	DSB	ECN bits
Syn	0 0	1 1
Syn-Ack	0 0	0 1
Ack	0 1	0 0
No Congestion	0 1	0 0
No Congestion	1 0	0 0
Congestion	1 1	0 0
Receiver Response	1 1	0 1
Sender Response	1 1	1 1

TCP Options

0 End of Options List
1 No Operation (NOP, Pad)
2 Maximum segment size
3 Window Scale
4 Selective ACK ok
8 Timestamp

Checksum

Checksum of entire TCP segment and pseudo header (parts of IP header)

Offset

Number of 32-bit words in TCP header, minimum value of 5. Multiply by 4 to get byte count.

RFC 793

Please refer to RFC 793 for the complete Transmission Control Protocol (TCP) Specification.

Figure 2: Source: https://www.nmap.org/

3.2 Packet Encapsulation/Decapsulation

I really felt the below illustration explained the context of packet encapsulation really well. The packet with a standard header is contained within the other packets that have additional headers detailing the locations in which the packet must travel.

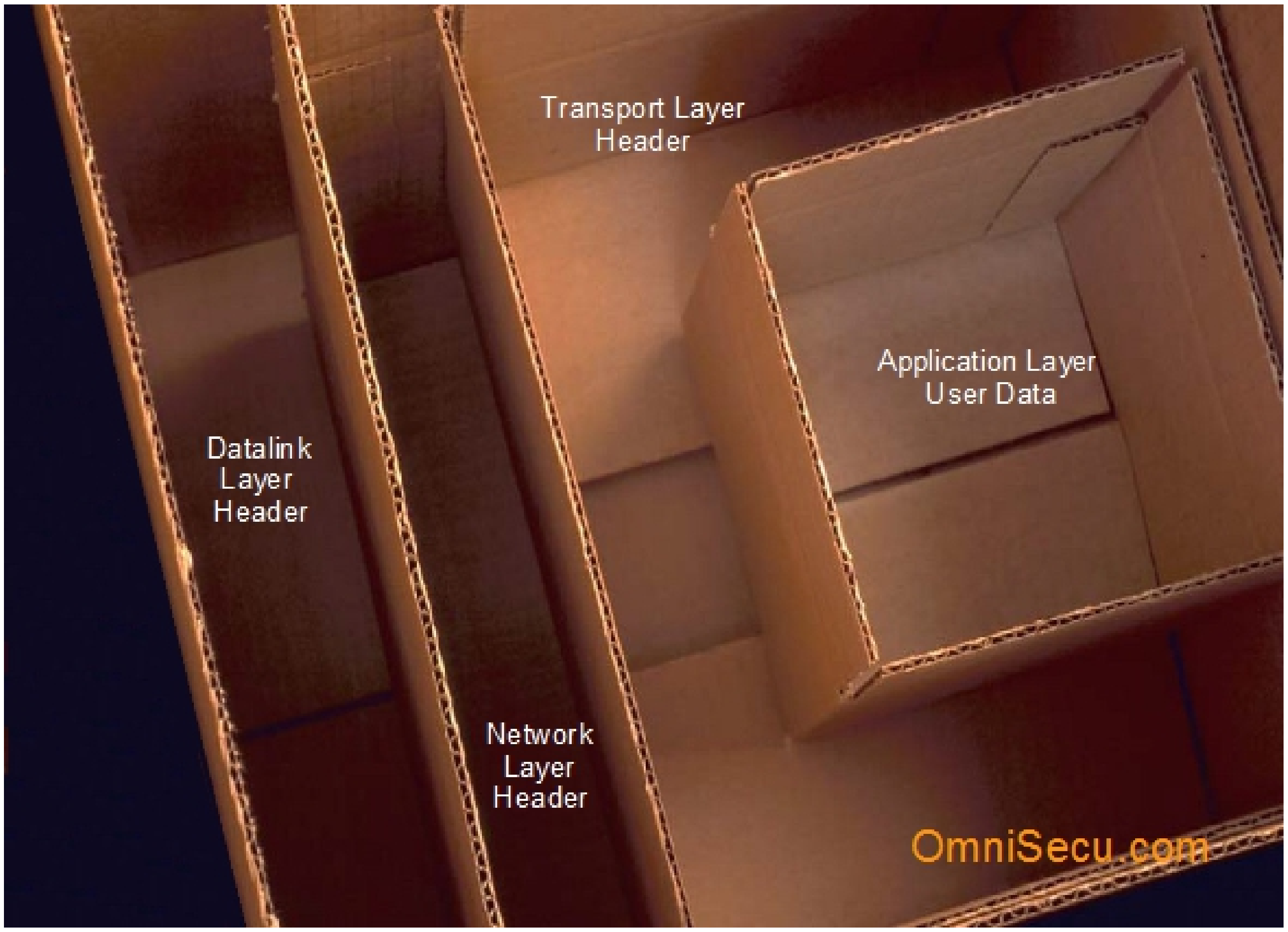


Figure 3: Example of Packet Encapsulation - Source: omnisecu.com

In my assignment packet encapsulation is used to take the packet from the client with a destination port of the server and add an additional header, so the packet now has another header ontop of the previous one with a distination port of the gateway.

In the gateway packet decapsulation is used, to remove the unnecessary header from the packet, after this is complete the packet is then sent on to the server from the gateway.

The process of packet encapsulation is then repeated in the server where it is setup to deliever to the gateway first and then the inner packet contains the destination as the correct corresponding client's port.

Packet decapsulation is then used once again in order to remove the additional header from the packet when a packet has been sent from the server to the gateway. This is then forwarded on to the correct corresponding client. This is all the cases in which packet encapsulation and decapsulation is used in this assignment.

3.3 Stop and Wait Protocol

I felt this diagram explained the process of Stop and Wait Protocols very well. The data is sent, if an acknowledgement packet is not received within the given timeframe the packet is then resent.

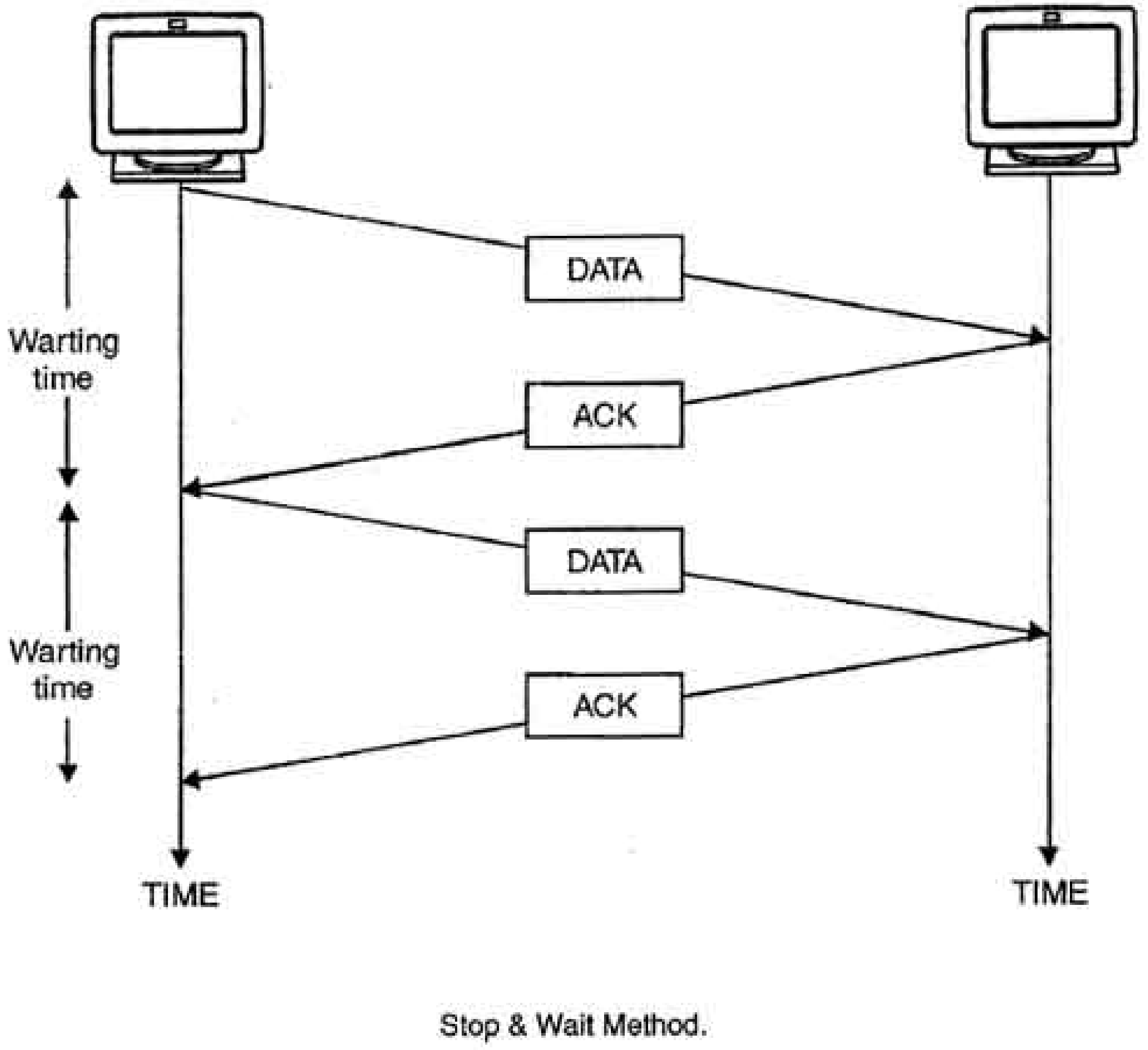


Figure 4: Diagram of Stop and Wait Protocol - Source ecomputernotes.com

My assignment implemented this process in order to ensure the packets are being delievered correctly to the server, this was really vital as within ensuring a packet was delievered correctly, I wouldn't have been able to implement the RSA encryption as I would be risking has the public keys been delievered safely. This would have been especially worrying if this was done on a larger scale outside of just being in a local network less likely to experience that most congestion.

3.4 Checksum and Sequence Numbers

Checksum and Sequence numbers are two features that ensure the correct packet has been send and the correct information has been transmitting successfully in that packet.

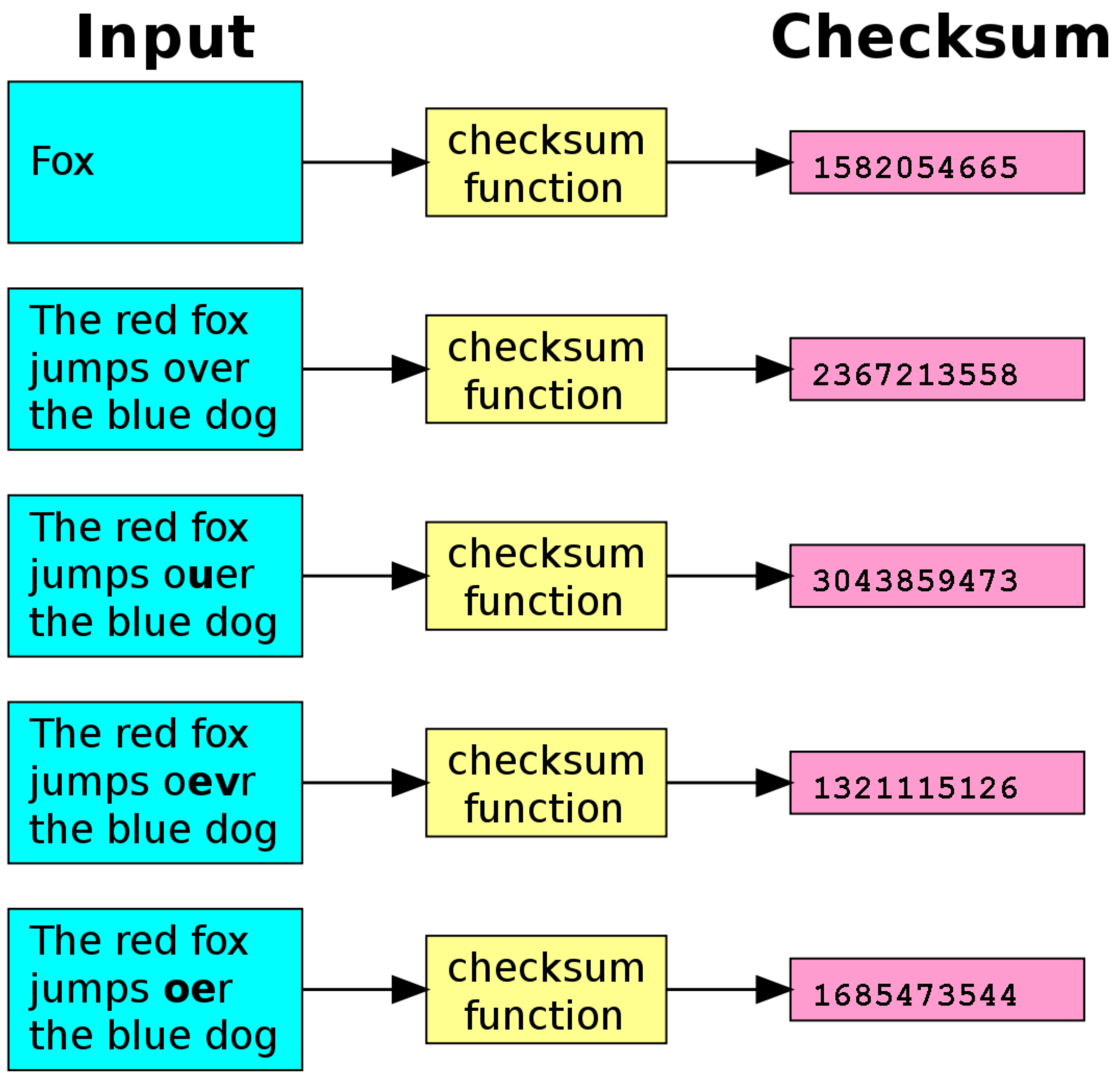


Figure 5: Example of Checksum - Source wikipedia.org

A checksum value is based on the input it receives. In this assignment the input is the packet content, the corresponding value of the checksum is then added to the header, this is used another method to ensure that all the bits have been transmitting correctly.
Sequence numbers are very simiply incrementing numbers depending on the packet being sent to ensure the correct order of packets.

3.5 Scalable - Multiple Clients and Multiple Servers

By implementing a useful header system, having the server store the ports with corresponding public keys and sequence numbers that it communicated with, packet encapsulating and a gateway that is completely dynamic in the addresses it sends packets to, this means that this code could easily be run for multiple clients and multiple servers, the only changes that would be necessary would be to change the server port number. The rest of the code will function completely without any changes.

4 RSA Encryption

4.1 Background

After getting experience snooping on packets during the first week of lectures and seeing how snooping can be done in class. I really wanted to try and prevent the data being transmitted in this network from being snooped on. This had to be done through RSA Encryption.

4.2 Implementation

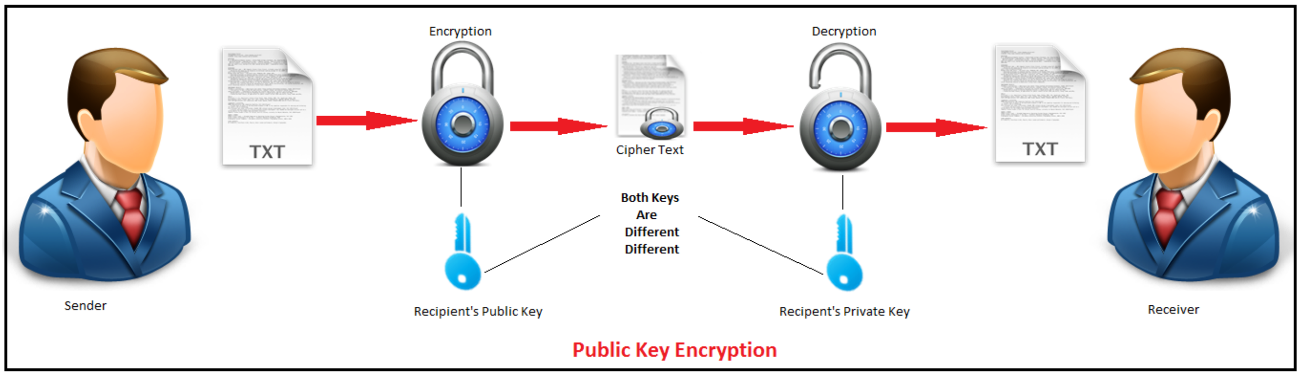


Figure 6: Example of Public - Private Key RSA Encryption - Source: c-sharpcorner.com

In order for RSA encryption to be implemented in this assignment, both the server and the client had to generate a key pair. The public key from this generated keypair had to be exchanged on both the client's and the server's end.

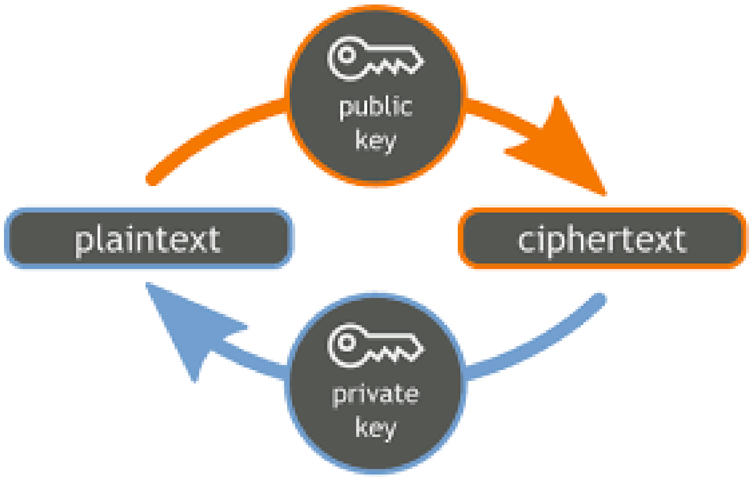
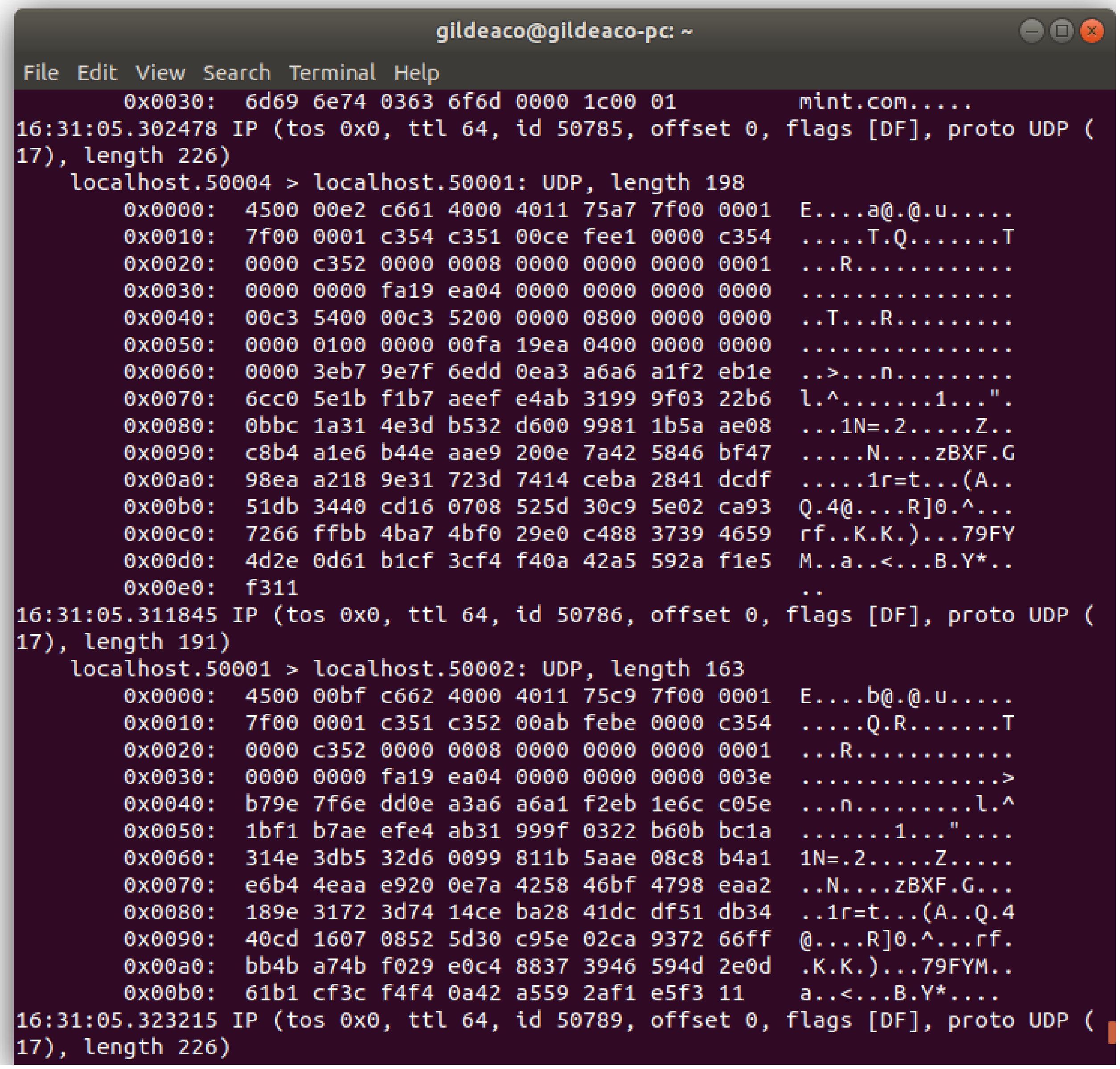


Figure 7: Example of Public - Private Key on Ciphertext to Plaintext- Source: latesthackingnews.com

During RSA encryption, the data is encrypted using the other person's public key. This data can only be unlocked using the other person's private key. As shown in the above diagram.

After the public keys have been exchanged any message for the server is encrypted using the server's public key and any message for the client is encrypted using the client's public key. This ensures the data is safe and secure and cannot be easily snooped on when it is being transmitted across the network, through the gateway.

4.3 Result of using RSA Encryption



```
        0x0030:  6d69 6e74 0363 6f6d 0000 1c00 01        mint.com.....
16:31:05.302478 IP (tos 0x0, ttl 64, id 50785, offset 0, flags [DF], proto UDP (
17), length 226)
    localhost.50004 > localhost.50001: UDP, length 198
        0x0000:  4500 00e2 c661 4000 4011 75a7 7f00 0001  E....a@.@.u.....
        0x0010:  7f00 0001 c354 c351 00ce fee1 0000 c354  .....T.Q.......T
        0x0020:  0000 c352 0000 0008 0000 0000 0000 0001  ...R............
        0x0030:  0000 0000 fa19 ea04 0000 0000 0000 0000  ................
        0x0040:  00c3 5400 00c3 5200 0000 0800 0000 0000  ..T...R.........
        0x0050:  0000 0100 0000 00fa 19ea 0400 0000 0000  ................
        0x0060:  0000 3eb7 9e7f 6edd 0ea3 a6a6 a1f2 eb1e  ..>...n.........
        0x0070:  6cc0 5e1b f1b7 aeef e4ab 3199 9f03 22b6  l.^.......1...".
        0x0080:  0bbc 1a31 4e3d b532 d600 9981 1b5a ae08  ...1N=.2.....Z..
        0x0090:  c8b4 a1e6 b44e aae9 200e 7a42 5846 bf47  .....N....zBXF.G
        0x00a0:  98ea a218 9e31 723d 7414 ceba 2841 dcdf  .....1r=t...(A..
        0x00b0:  51db 3440 cd16 0708 525d 30c9 5e02 ca93  Q.4@....R]0.^...
        0x00c0:  7266 ffbb 4ba7 4bf0 29e0 c488 3739 4659  rf..K.K.)...79FY
        0x00d0:  4d2e 0d61 b1cf 3cf4 f40a 42a5 592a f1e5  M..a..<...B.Y*..
        0x00e0:  f311                                     ..
16:31:05.311845 IP (tos 0x0, ttl 64, id 50786, offset 0, flags [DF], proto UDP (
17), length 191)
    localhost.50001 > localhost.50002: UDP, length 163
        0x0000:  4500 00bf c662 4000 4011 75c9 7f00 0001  E....b@.@.u.....
        0x0010:  7f00 0001 c351 c352 00ab febe 0000 c354  .....Q.R.......T
        0x0020:  0000 c352 0000 0008 0000 0000 0000 0001  ...R............
        0x0030:  0000 0000 fa19 ea04 0000 0000 0000 003e  ...............>
        0x0040:  b79e 7f6e dd0e a3a6 a6a1 f2eb 1e6c c05e  ...n.........l.^
        0x0050:  1bf1 b7ae efe4 ab31 999f 0322 b60b bc1a  .......1..."....
        0x0060:  314e 3db5 32d6 0099 811b 5aae 08c8 b4a1  1N=.2.....Z.....
        0x0070:  e6b4 4eaa e920 0e7a 4258 46bf 4798 eaa2  ..N....zBXF.G...
        0x0080:  189e 3172 3d74 14ce ba28 41dc df51 db34  ..1r=t...(A..Q.4
        0x0090:  40cd 1607 0852 5d30 c95e 02ca 9372 66ff  @....R]0.^...rf.
        0x00a0:  bb4b a74b f029 e0c4 8837 3946 594d 2e0d  .K.K.)...79FYM..
        0x00b0:  61b1 cf3c f4f4 0a42 a559 2af1 e5f3 11    a..<...B.Y*....
16:31:05.323215 IP (tos 0x0, ttl 64, id 50789, offset 0, flags [DF], proto UDP (
17), length 226)
```

Figure 8: Encrypted Output of what is being sent from the Client to the Gateway then to the Server

I am really delighted with my implementation of RSA Encryption, it ensures that if you try to snoop on the content of the packet in with tcpdump or in a program such as wireshark you are unable to see the content of the message as you cannot decrypt the messages without the private key.

This really added a really high level of sophistication to my assignment as it ensured the client/s and server/s had total privacy from people snooping on the content of the traffic between the two. Only the client and the server can easily access their corresponding data.

5 Assignment Reflection

5.1 Advantages of Implementation

- Multi-clients can be added easily, simply by running the client code, a new free port will be found an assigned.
- Code is scalable for multiple clients and servers to operate using the same gateway.
- RSA Encryption ensures privacy preventing anyone besides the client and server from snooping on the content of the packets they are exchanging.
- Stop and Wait Protocol works as attending, ensuring packets get delievered correctly eventually.

5.2 Disadvantages of Implementation

- Not an exact standard header was used. I didn't follow albeit exactly to the TCP protocol, I did a shorter version only using the information applicable for this assignment, I could have implemented the full TCP protocol however I felt this was slightly unnecessary and would have just been a lot of empty bytes near the start of each packet.
- RSA Encryption is an incredible addition for security in this network, I guess, to a degree, it could be slighty unnecessary if this network is just being used to transmit non-sensitive data.
- This implementation has been designed for using within the local network, I could have extended it further in my header to allow for different IPs to be added, this would have been an easy additional feature to add although would make my code adhere to not being as similiar to TCP.
- Although I feel I covered a lot, there is always the potential for more bug-fixes and maintenance.

5.3 Time Spent

In total I spent around **16-22 working hours** on this assignment's code and an additional **4-5 working hours** on this report.

I also spent a significant amount of time researching the best approach I could have to certain topics such as the RSA encryption and re-reading the notes for additional assistance.

5.4 What I have learnt...

I really enjoyed working on this assignment as I got a much more practical sense of the topics discussed in the lecture, I also got to do some additional learning regarding RSA encryption which I found extremely useful to learn.

Topics I learnt/got to implement throughout the course of this assignment are:

- Allocation and use of sockets.
- Implementation of ports
- Packet Encapsulation
- Threading
- Acknowledgements
- ARQs - Stop and Wait
- Sequence Numbers
- Checksum
- Headers
- TCP
- UDP
- Key Generation - Public and Private Keys
- RSA Encryption