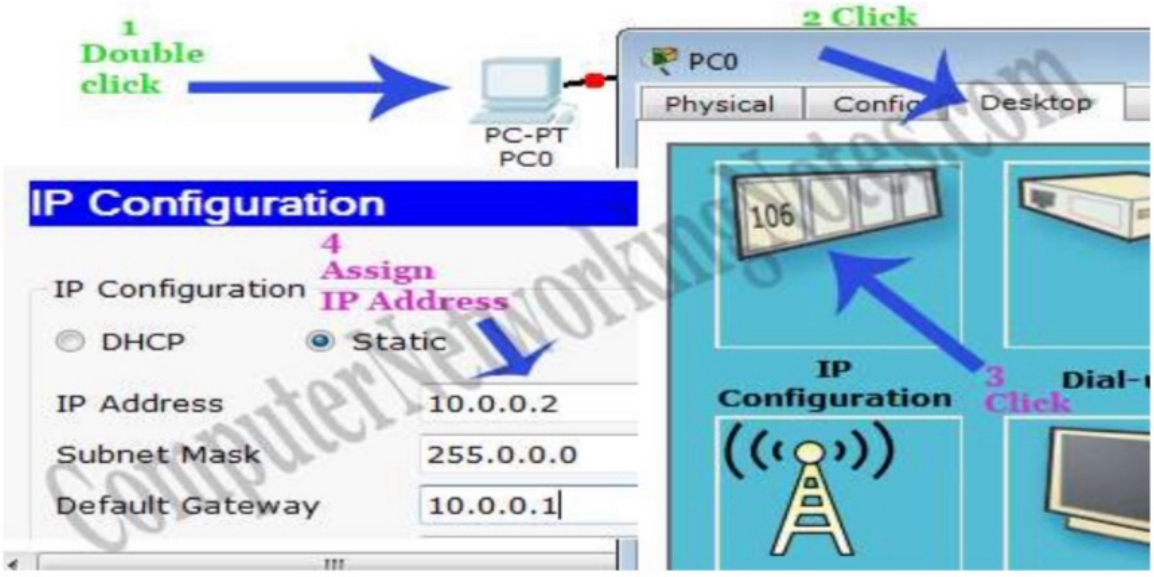
Initial IP configuration

Device	Interface	IP Configuration	Connected with
PC0	Fast Ethernet	10.0.0.2/8	Router0's Fa0/1
Router0	Fa0/1	10.0.0.1/8	PC0's Fast Ethernet
Router0	S0/0/1	192.168.1.254/30	Router2's S0/0/1
Router0	S0/0/0	192.168.1.249/30	Router1's S0/0/0
Router1	S0/0/0	192.168.1.250/30	Router0's S0/0/0
Router1	S0/0/1	192.168.1.246/30	Router2's S0/0/0
Router2	S0/0/0	192.168.1.245/30	Router1's S0/0/1
Router2	S0/0/1	192.168.1.253/30	Router0's S0/0/1
Router2	Fa0/1	20.0.0.1/30	PC1's Fast Ethernet
PC1	Fast Ethernet	20.0.0.2/30	Router2's Fa0/1

Assign IP address to PCs

Double click PC0 and click Desktop menu item and click IP Configuration. Assign IP address 10.0.0.2/8 to PC0.



Repeat same process for PC1 and assign IP address 20.0.0.2/8.

Assign IP address to interfaces of routers

Double click **Router0** and click **CLI** and press **Enter key** to access the command prompt of **Router0**.

ip address ***10.0.0.1 255.0.0.0*** command will assign IP address to interface.

no shutdown command will bring the interface up.

exit command is used to return in global configuration mode.

Serial interface needs two additional parameters **clock rate** and **bandwidth**. Every serial cable has two ends DTE and DCE. These parameters are always configured at DCE end.

We can use **show controllers interface** command from privilege mode to check the cable's end.

Router#show controllers serial 0/0/0

Interface Serial0/0/0

Hardware is PowerQUICC MPC860

DCE V.35, clock rate 2000000

[Output omitted]

Fourth line of output confirms that DCE end of serial cable is attached. If you see DTE here instead of DCE skip these parameters.

Now we have necessary information let's assign IP address to serial interface.

Router#configure terminal

Enter configuration commands, one per line. End with CNTL/Z.

Router(config)#interface serial 0/0/0

Router(config-if)#ip address 192.168.1.249 255.255.255.252

Router(config-if)#clock rate 64000

Router(config-if)#bandwidth 64

Router(config-if)#no shutdown

Router(config-if)#exit

Router(config)#interface serial 0/0/1

Router(config-if)#ip address 192.168.1.254 255.255.255.252

Router(config-if)#clock rate 64000

Router(config-if)#bandwidth 64

Router(config-if)#no shutdown

Router(config-if)#exit

Router(config)#

Router#configure terminal Command is used to enter in global configuration mode.

Router(config)#interface ***serial 0/0/0*** Command is used to enter in interface mode.

Router(config-if)#ip address ***192.168.1.249 255.255.255.252*** Command assigns IP address to interface. For serial link we usually use IP address from /30 subnet.

Router(config-if)#clock rate ***64000*** And **Router(config-if)#bandwidth** ***64*** In real life environment these parameters control the data flow between serial links and need to be set at service providers end. In lab environment we need not to worry about these values. We can use these values.

Router(config-if)#no shutdown Command brings interface up.

Router(config-if)#exit Command is used to return in global configuration mode.

We will use same commands to assign IP addresses on interfaces of remaining routers. We need to provided clock rate and bandwidth only on DCE side of serial interface. Following command will assign IP addresses on interface of Router1.

Router1.

Router>enable

Router#configure terminal

Enter configuration commands, one per line. End with CNTL/Z.

Router(config)#interface serial 0/0/0

Router(config-if)#ip address 192.168.1.250 255.255.255.252

Router(config-if)#no shutdown

Router(config-if)#exit

Router(config)#interface serial 0/0/1

Router(config-if)#ip address 192.168.1.246 255.255.255.252

Router(config-if)#clock rate 64000

Router(config-if)#bandwidth 64

Router(config-if)#no shutdown

Router(config-if)#exit

Use same commands to assign IP addresses on interfaces of Router2.

Router2

Router>enable

Router#configure terminal

Enter configuration commands, one per line. End with CNTL/Z.

Router(config)#interface fastEthernet 0/0

Router(config-if)#ip address 20.0.0.1 255.0.0.0

Router(config-if)#no shutdown

Router(config-if)#exit

Router(config)#interface serial 0/0/0

Router(config-if)#ip address 192.168.1.245 255.255.255.252

Router(config-if)#no shutdown

Router(config-if)#exit

Router(config)#interface serial 0/0/1

Router(config-if)#ip address 192.168.1.253 255.255.255.252

Router(config-if)#no shutdown

Router(config-if)#exit

Now routers have information about the networks that they have on their own interfaces. Routers will not exchange this information between them on their own. We need to implement RIP routing protocol that will insist them to share this information.

Configure RIP routing protocol

Configuration of RIP protocol is much easier than you think. It requires only two steps to configure the RIP routing.

- Enable RIP routing protocol from global configuration mode.
- Tell RIP routing protocol which networks you want to advertise.

Let's configure it in Router0

Router0

```
Router0(config)#router rip
Router0(config-router)# network 10.0.0.0
Router0(config-router)# network 192.168.1.252
Router0(config-router)# network 192.168.1.248
```

router rip command tell router to enable the RIP routing protocol.

network command allows us to specify the networks which we want to advertise. We only need to specify the networks which are directly connected with the router.

That's all we need to configure the RIP. Follow same steps on remaining routers.

Router1

Router1(config)#router rip

Router1(config-router)# network 192.168.1.244

Router1(config-router)# network 192.168.1.248

Router2

The first command enables EIGRP with the AS number 20. Since we used the AS number 20 on this router, this router will share routing information only with the routers that belong to the AS number 20.

The second command adds the network **10.0.0.0** ***255.0.0.0*** to the EIGRP operation. The subnet mask of the network **10.0.0.0*/8*** is *255.0.0.0*. To calculate the wildcard mask for this subnet mask, we subtracted if from 255.255.255.255.

255 .255 .255 .255
– 255 .0 .0 .0
---
0 .255 .255 .255

When we add a network, EIGRP checks the IP configuration of all interfaces and enables EIGRP operation on the matching interfaces. Since the IP configuration of the interface **F0/0** belongs to the network **10.0.0.0*/8***, EIGRP enables EIGRP operation on the interface F0/0.

*In network diagrams, we normally use slash notation to write the subnet mask. In slash notation, the subnet mask **255.0.0.0** is written as **/8** and the subnet mask **255.255.255.252** is written as **/30**.*

The third command adds the network **192.168.1.244** ***255.255.255.252*** **(/30)** to the EIGRP operation. Since this network is configured on the Serial interface **S0/0/0**, EIGRP enables EIGRP operation on the Serial interface **S0/0/0**.

The fourth command adds the network **192.168.1.0** ***255.255.255.252*** **(/30)** to the EIGRP operation. This network is configured on the **F0/1** interface. Thus, EIGRP enables the EIGRP operation on the **F0/1** interface.

Using the same commands in the same pattern, we can configure and enable EIGRP on the rest of the routers.

Router 1

```
Router(config)#router eigrp 20
Router(config-router)#network 192.168.1.244 0.0.0.3
Router(config-router)#network 192.168.1.248 0.0.0.3
Router(config-router)#
```

Router 2

Router(config)#router eigrp 20

Router(config-router)#network 192.168.1.248 0.0.0.3

Router(config-router)#network 192.168.1.252 0.0.0.3

Router(config-router)#

Router 3

Router(config)#router eigrp 20

Router(config-router)#network 192.168.1.8 0.0.0.3

Router(config-router)#network 192.168.1.4 0.0.0.3

Router(config-router)#

Router 3

Router(config)#router eigrp 20

Router(config-router)#network 192.168.1.4 0.0.0.3

Router(config-router)#network 192.168.1.0 0.0.0.3

Router(config-router)#

Router5

Router(config)#router eigrp 20

Router(config-router)#network 20.0.0.0 0.255.255.255

Router(config-router)#network 192.168.1.252 0.0.0.3

Router(config-router)#network 192.168.1.8 0.0.0.3

Router(config-router)#

The following image shows how to run the above commands on routers.

Router0

```
Router>enable
Router#configure terminal
Enter configuration commands, one per line.  End wi
Router(config)#router eigrp 20
Router(config-router)#network 10.0.0.0 0.0.0.255
Router(config-router)#network 192.168.1.244 0.0.0.3
Router(config-router)#network 192.168.1.0 0.0.0.3
```

Router1

```
Router>enable
Router#configure terminal
Enter configuration commands, one per line.  End witl
Router(config)#router eigrp 20
Router(config-router)#network 192.168.1.244 0.0.0.3
Router(config-router)#network 192.168.1.248 0.0.0.3
Router(config-router)#
```

Router2

```
Router>enable
Router#configure terminal
Enter configuration commands, one per line.  End wit
Router(config)#router eigrp 20
Router(config-router)#network 192.168.1.248 0.0.0.3
Router(config-router)#network 192.168.1.252 0.0.0.3
Router(config-router)#exit
Router(config)#
```

Router3

```
Router>enable
Router#configure terminal
Enter configuration commands, one per line.  End wit
Router(config)#router eigrp 20
Router(config-router)#network 192.168.1.8 0.0.0.3
Router(config-router)#network 192.168.1.4 0.0.0.3
Router(config-router)#exit
Router(config)#
```

Router4

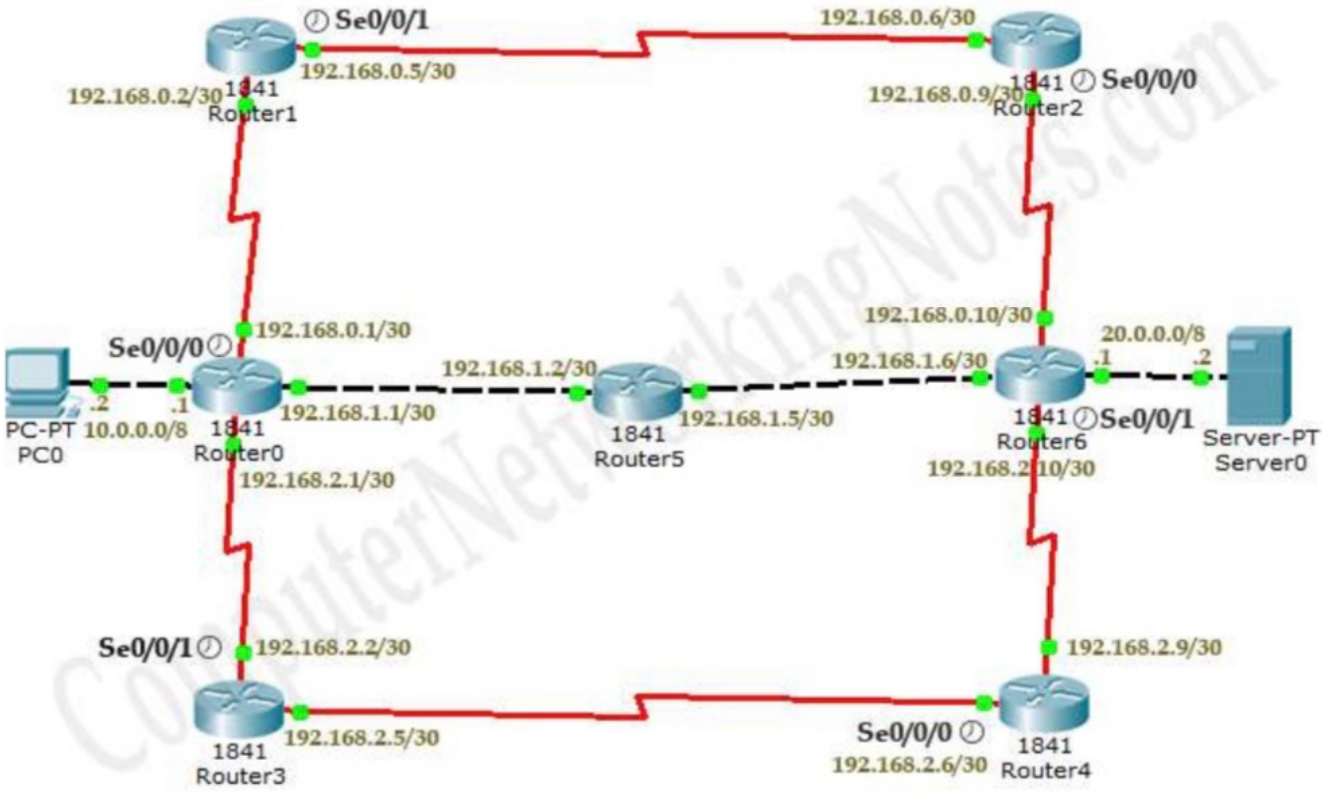
```
Router>enable
Router#configure terminal
Enter configuration commands, one per line.  End with
Router(config)#router eigrp 20
Router(config-router)#network 192.168.1.4 0.0.0.3
Router(config-router)#network 192.168.1.0 0.0.0.3
Router(config-router)#
Router(config-router)#
Router(config-router)#
```

Router5

```
Router>enable
Router#configure terminal
Enter configuration commands, one per line.  End wit
Router(config)#router eigrp 20
Router(config-router)#network 20.0.0.0 0.255.255.255
Router(config-router)#network 192.168.1.252 0.0.0.3
Router(config-router)#network 192.168.1.8 0.0.0.3
Router(config-router)#exit
Router(config)#
```

OSPF Configuration step by step procedure

For demonstration we will use packet tracer network simulator software. You can use real Cisco devices or any other network simulator software for following this guide.



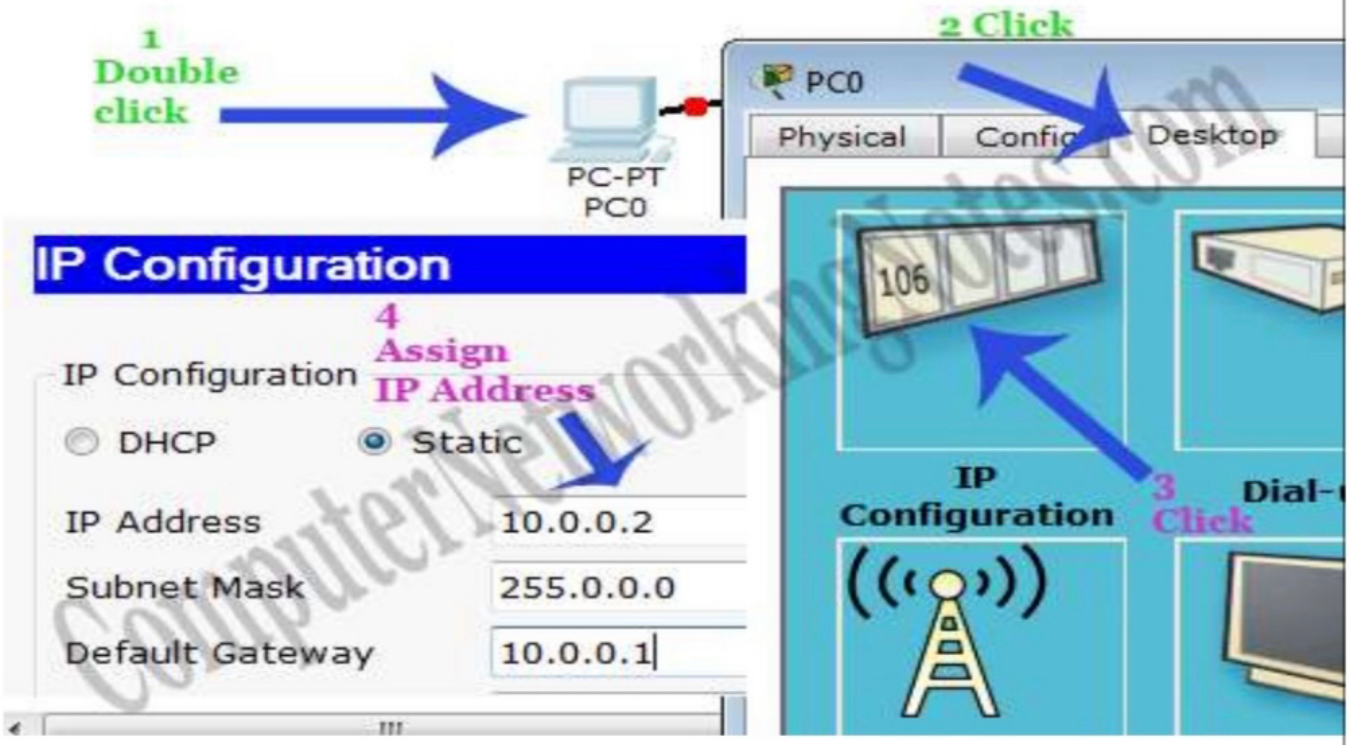
Initial IP Configuration

Device	Interface	IP Configuration	Connected with
PC0	Fa0/0	10.0.0.2/8	Router0's Fa0/0
Router0	Fa0/0	10.0.0.1/8	PC0's Fa0/0
Router0	Fa0/1	192.168.1.1/30	Router5's Fa0/1
Router5	Fa0/1	192.168.1.2/30	Router0's Fa0/1
Router5	Fa0/0	192.168.1.5/30	Router6's F0/0
Router6	Fa0/0	192.168.1.6/30	Router5's Fa0/0
Router6	Fa0/1	20.0.0.1/8	Server0's Fa0/0
Server0	Fa0/0	20.0.0.2/8	Router6's Fa0/1
Router0	Serial 0/0/0 (DCE)	192.168.0.1/30	Router1's Se0/0/0
Router1	Serial 0/0/0	192.168.0.2/30	Router0's Se0/0/0
Router1	Serial 0/0/1 (DCE)	192.168.0.5/30	Router2's Se0/0/1

Device	Interface	IP Configuration	Connected with
Router2	Serial0/0/1	192.168.0.6/30	Router1's Se0/0/1
Router2	Serial 0/0/0 (DCE)	192.168.0.9/30	Router6's Se0/0/0
Router6	Serial 0/0/0	192.168.0.10/30	Router2's Se0/0/0
Router0	Serial 0/0/1	192.168.2.1/30	Router3's Se0/0/1
Router3	Serial 0/0/1 (DCE)	192.168.2.2/30	Router0's Se0/0/1
Router3	Serial 0/0/0	192.168.2.5/30	Router4's Se0/0/0
Router4	Serial 0/0/0 (DCE)	192.68.2.6/30	Router3's Se0/0/0
Router4	Serial 0/0/1	192.168.2.9/30	Router6's Se0/0/1
Router6	Serial0/0/1 (DCE)	192.168.2.10/30	Router4's Se0/0/1

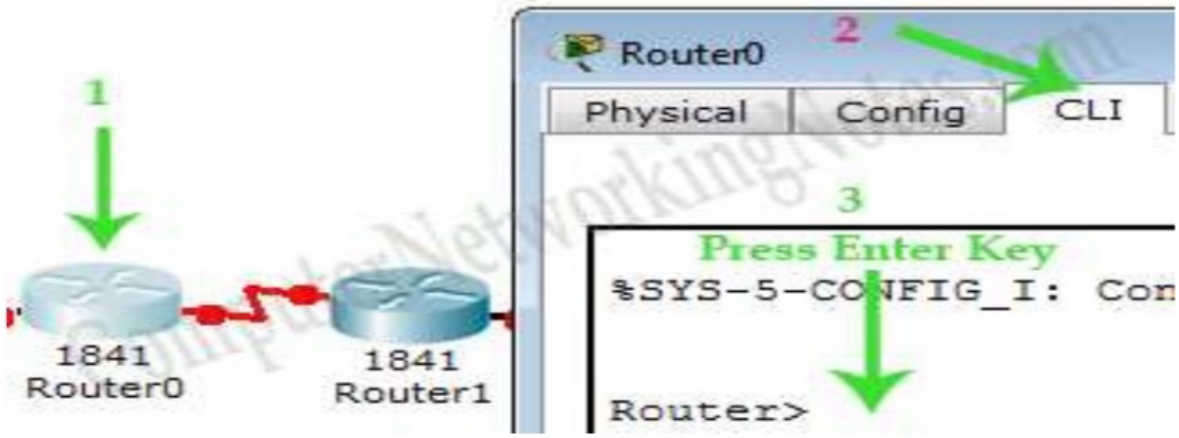
Assign IP address to PC

Double click PC0 and click Desktop menu item and click IP Configuration. Assign IP address *10.0.0.2/8* to PC0.



Assign IP address to interfaces of routers

Double click Router0 and click CLI and press Enter key to access the command prompt of Router0.



Four interfaces FastEthernet0/0, FastEthernet0/1, Serial 0/0/0 and Serial0/0/1 of Router0 are used in this topology. By default, interfaces on router are remain administratively down during the start up. We need to configure IP address and other parameters on interfaces before we could actually use them for routing. Interface mode is used to assign the IP address and other parameters. Interface mode can be accessed from global configuration mode. Following commands are used to access the global configuration mode.

Router>enable

Router# configure terminal

Enter configuration commands, one per line. End with CNTL/Z.

Router(config)#

From global configuration mode we can enter in interface mode. From there we can configure the interface. Following commands will assign IP address on FastEthernet0/0 and ***FastEthernet0/1.***

```
Router(config)#interface fastEthernet 0/0
Router(config-if)#ip address 10.0.0.1 255.0.0.0
Router(config-if)#no shutdown
Router(config-if)#exit
Router(config)#interface fastEthernet 0/1
Router(config-if)#ip address 192.168.1.1 255.255.255.252
Router(config-if)#no shutdown
Router(config-if)#exit
```

interface fastEthernet 0/0 command is used to enter in interface mode.

ip address 10.0.0.1 255.0.0.0 command would assign IP address to interface.

no shutdown command would bring the interface up.

exit command is used to return in global configuration mode.

Serial interface needs two additional parameters **clock rate** and **bandwidth**. Every serial cable has two ends DTE and DCE. These parameters are always configured at DCE end.

We can use **show controllers *interface*** command from privilege mode to check the cable's end.

```
Router#show controllers serial 0/0/0
Interface Serial0/0/0
Hardware is PowerQUICC MPC860
DCE V.35, clock rate 2000000
[Output omitted]
```

Fourth line of output confirms that DCE end of serial cable is attached. If you see DTE here instead of DCE skip these parameters.

Now we have necessary information let's assign IP address to serial interfaces.

```
Router# configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#interface serial 0/0/0
Router(config-if)#ip address 192.168.0.1  255.255.255.252
Router(config-if)#clock rate 64000
Router(config-if)#bandwidth 64
Router(config-if)#no shutdown
Router(config-if)#exit
Router(config)#interface serial 0/0/1
Router(config-if)#ip address 192.168.2.1  255.255.255.252
Router(config-if)#no shutdown
Router(config-if)#exit
```

Router#configure terminal Command is used to enter in global configuration mode.

Router(config)#interface serial 0/0/0 Command is used to enter in interface mode.

Router(config-if)#ip address 192.168.0.1 255.255.255.252 Command assigns IP address to interface. For serial link we usually use IP address from /30 subnet.

Router(config-if)#clock rate 64000 In real life environment this parameter controls the data flow between serial links and need to be set at service provider's end. In lab environment we need not to worry about this value. We can use any valid clock rate here.

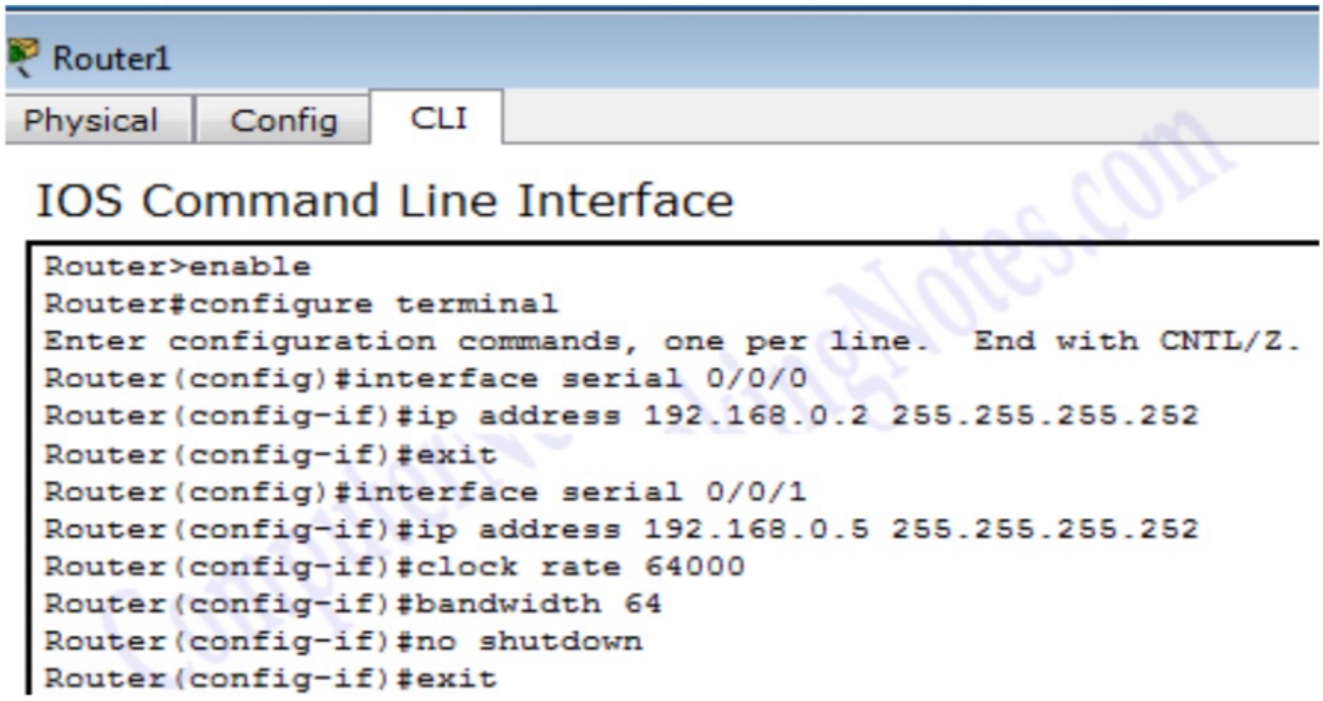
Router(config-if)#bandwidth 64 Bandwidth works as an influencer. It is used to influence the metric calculation of OSPF or any other routing protocol which uses bandwidth parameter in route selection process. Serial interface has default bandwidth of 1544Kbps. To explain, how bandwidth influence route selection process we will configure (64Kbps) bandwidth on three serial DCE interfaces of our network; R0's Se0/0/0, R1's Se0/0/1 and R2's Se0/0/0.

Router(config-if)#no shutdown Command brings interface up.

Router(config-if)#exit Command is used to return in global configuration mode.

We will use same commands to assign IP addresses on interfaces of remaining routers.

Router1




```
Router>enable
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#interface serial 0/0/0
Router(config-if)#ip address 192.168.0.2 255.255.255.252
Router(config-if)#exit
Router(config)#interface serial 0/0/1
Router(config-if)#ip address 192.168.0.5 255.255.255.252
Router(config-if)#clock rate 64000
Router(config-if)#bandwidth 64
Router(config-if)#no shutdown
Router(config-if)#exit
```

Router2

```
Router>enable
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#interface serial 0/0/0
Router(config-if)#ip address 192.168.0.9 255.255.255.252
Router(config-if)#clock rate 64000
Router(config-if)#bandwidth 64
Router(config-if)#no shutdown
Router(config-if)#exit
Router(config-if)#
Router(config)#interface serial 0/0/1
Router(config-if)#ip address 192.168.0.6 255.255.255.252
Router(config-if)#no shutdown
Router(config-if)#exit
Router(config)#
```

As I mention earlier, serial interface has a default bandwidth of 1544Kbps. If we don't assign any custom bandwidth, router would use default bandwidth. To see this feature in action we will not assign bandwidth on remaining routers.

Router6

Physical | Config | CLI

IOS Command Line Interface

```
Router>enable
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#interface serial 0/0/0
Router(config-if)#ip address 192.168.0.10 255.255.255.252
Router(config-if)#no shutdown
Router(config-if)#exit
Router(config)#interface serial 0/0/1
Router(config-if)#ip address 192.168.2.10 255.255.255.252
Router(config-if)#clock rate 64000
Router(config-if)#no shutdown
Router(config-if)#exit
Router(config)#interface fastethernet 0/0
Router(config-if)#ip address 192.168.1.6 255.255.255.252
Router(config-if)#no shutdown
Router(config-if)#exit
Router(config)#interface fastethernet 0/1
Router(config-if)#ip address 20.0.0.1 255.0.0.0
Router(config-if)#no shutdown
Router(config-if)#exit
```

Router5

Router5

Physical | Config | CLI

IOS Command Line Interface

```
Router>enable
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#interface fastethernet 0/0
Router(config-if)#ip address 192.168.1.5 255.255.255.252
Router(config-if)#no shutdown
Router(config-if)#exit
Router(config)#interface fastethernet 0/1
Router(config-if)#ip address 192.168.1.2 255.255.255.252
Router(config-if)#no shutdown
Router(config-if)#exit
Router(config)#
```

Router3

```
Router>enable
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#interface serial 0/0/0
Router(config-if)#ip address 192.168.2.5 255.255.255.252
Router(config-if)#no shutdown
Router(config-if)#exit
Router(config)#interface serial 0/0/1
Router(config-if)#ip address 192.168.2.2 255.255.255.252
Router(config-if)#clock rate 64000
Router(config-if)#no shutdown
Router(config-if)#exit
Router(config)#
```

Router4

Physical | Config | CLI

IOS Command Line Interface

```
Router>enable
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#interface serial 0/0/0
Router(config-if)#ip address 192.168.0.10 255.255.255.252
Router(config-if)#no shutdown
Router(config-if)#exit
Router(config)#interface serial 0/0/1
Router(config-if)#ip address 192.168.2.10 255.255.255.252
Router(config-if)#clock rate 64000
Router(config-if)#no shutdown
Router(config-if)#exit
Router(config)#interface fastethernet 0/0
Router(config-if)#ip address 192.168.1.6 255.255.255.252
Router(config-if)#no shutdown
Router(config-if)#exit
Router(config)#interface fastethernet 0/1
Router(config-if)#ip address 20.0.0.1 255.0.0.0
Router(config-if)#no shutdown
Router(config-if)#exit
```

we have finished our half journey. Now routers have information about the networks that they have on their own interfaces. Routers will not exchange this information between them on their own. We need to implement OSPF routing protocol that will insist them to share this information.

Configure OSPF routing protocol

Enabling OSPF is a two steps process:-

- Enable OSPF routing protocol from global configuration mode.
- Tell OSPF which interfaces we want to include.

For these steps following commands are used respectively.

Router(config)# router ospf process_ID

Router(config-router)# network IP_network_# [wild card mask] Area Number area number

Router(config)# router ospf process ID

This command will enable OSPF routing protocol in router. Process ID is a positive integer. We can use any number from 1 to 65,535. Process ID is locally significant. We can run multiple OSPF process on same router. Process ID is used to differentiate between them. Process ID need not to match on all routers.

Router(config-router)# network IP_network_# [wildcard_mask] area [area number]

Network command allows us to specify the interfaces which we want to include in OSPF process. This command accepts three arguments network number, wildcard mask and area number.

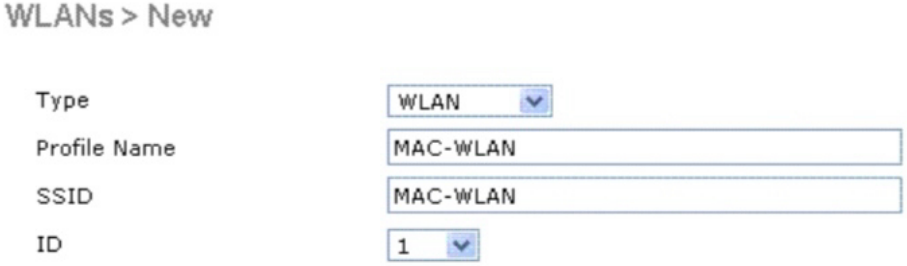
already configured for basic operation and that the LAPs are registered to the WLC. If you are a new user trying to set up the WLC for basic operation with LAPs, refer to Lightweight AP (LAP) Registration to a Wireless LAN Controller (WLC).

Note: There is no special configuration needed on the wireless client in order to support MAC authentication.

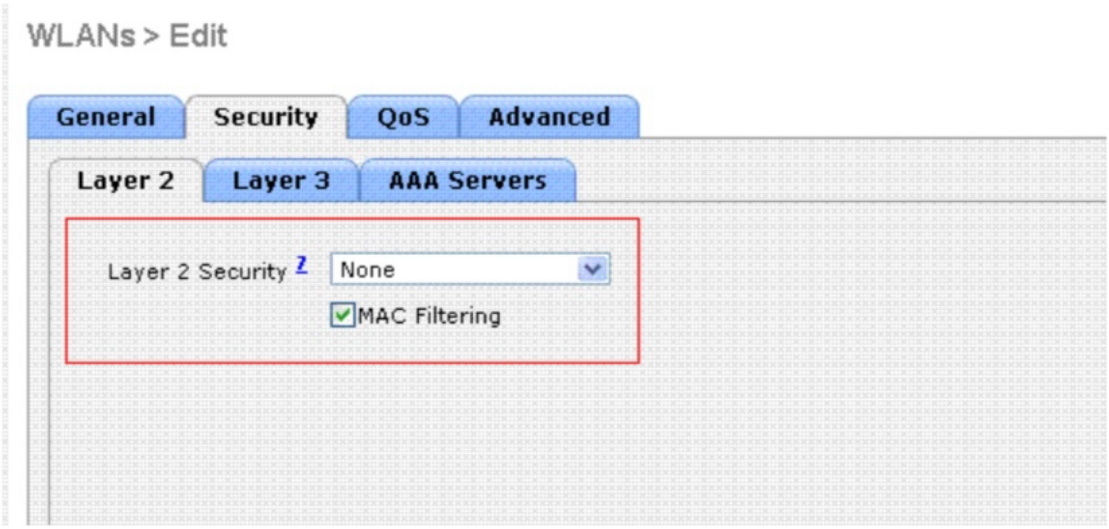
Configure a WLAN and Enable MAC Filtering

Complete these steps in order to configure a WLAN with MAC filtering:

1. Click **WLANs** from the controller GUI in order to create a WLAN.
   The WLANs window appears. This window lists the WLANs configured on the controller.
2. Click **New** in order to configure a new WLAN.
   In this example, the WLAN is named *MAC-WLAN* and the WLAN ID is *1*.



3. Click **Apply**.
4. In the WLAN > Edit window, define the parameters specific to the WLAN.



. Under Security Policies > Layer 2 Security, check the **MAC Filtering** check box.
This enables MAC authentication for the WLAN.

a. Under General Policies > Interface Name, select the interface to which the WLAN is mapped.

   In this example, the WLAN is mapped to the management interface.

b. Select the other parameters, which depend on the design requirements of the WLAN.

c. Click **Apply**.

The next step is to configure the local database on the WLC with the client MAC addresses.

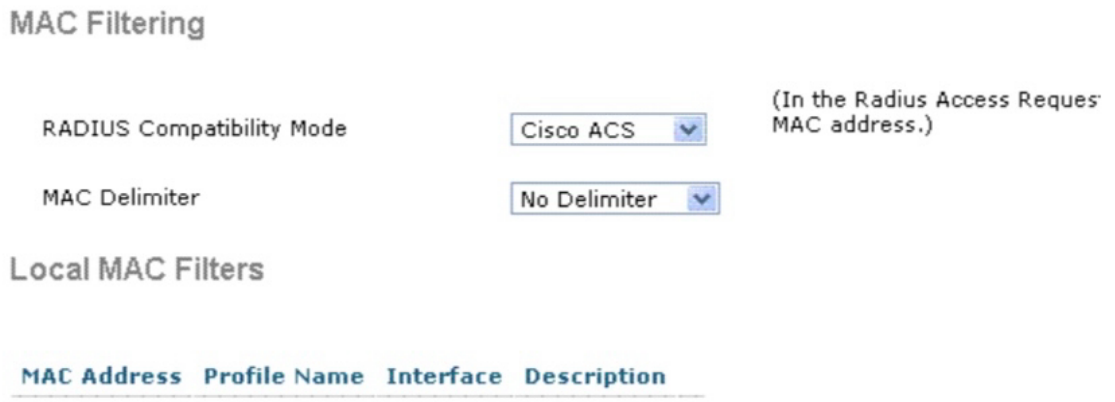
Refer to VLANs on Wireless LAN Controllers Configuration Example for information on how to configure dynamic interfaces (VLANs) on WLCs.

Configure the Local Database on the WLC with Client MAC Addresses

Complete these steps in order to configure the local database with a client MAC address on the WLC:

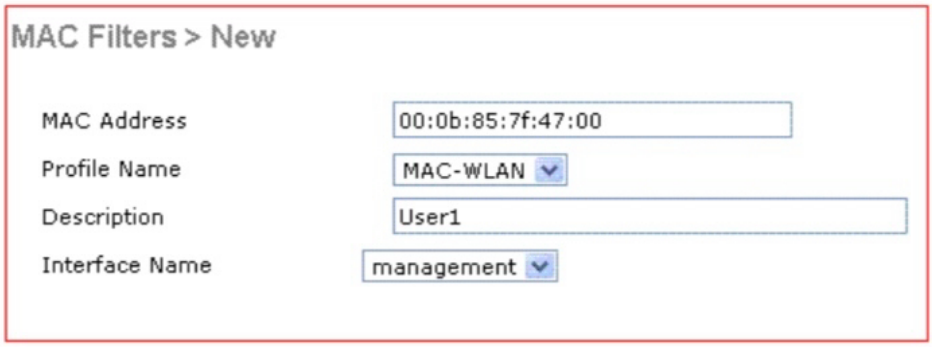
1. Click **Security** from the controller GUI, and then click **MAC Filtering** from the left side menu.

   The MAC Filtering window appears.



2. Click **New** in order to create a local database MAC address entry on the WLC.
3. In the MAC Filters > New window, enter the MAC address, Profile Name, Description and the Interface Name for the client.

   Here is an example:



4. Click **Apply**.
5. Repeat steps 2-4 in order to add more clients to the local database.

   Now, when clients connect to this WLAN, the WLC validates the clients MAC address against the local database and if the validation is successful, the client is granted access to the network.

   Note: In this example, only a MAC address filter without any other Layer 2 Security mechanism was used. Cisco recommends that MAC address authentication should be used along with other Layer 2 or Layer 3 security methods. It is not advisable to use only MAC address authentication to secure your WLAN network because it does not provide a strong security mechanism.

Compiling:

gcc client.c -o client gcc server.c -o server

Output:

Client:Hello message sent Hello from server Server:Hello from client Hello message sent

UDP Client , UDP Server

UDP Socket

This article describes how to write a simple echo server and client using udp sockets in C on Linux/Unix platform. UDP sockets or Datagram sockets are different from the TCP sockets in a number of ways. The most important difference is that UDP sockets are not connection oriented. More technically speaking, a UDP server does not accept connections and a udp client does not connect to server.

The server will bind and then directly receive data and the client shall directly send the data.

ECHO Server

So lets first make a very simple ECHO server with UDP socket. The flow of the code would be

socket() -> bind() -> recvfrom() -> sendto()

C code

```
/*
Simple udp server
*/
#include<stdio.h> //printf
#include<string.h> //memset
#include<stdlib.h> //exit(0);
#include<arpa/inet.h>
#include<sys/socket.h>
#define BUFLEN 512 //Max length of buffer
#define PORT 8888 //The port on which to listen for incoming data
void die(char *s)
```

```
{
perror(s);
exit(1);
}
int main(void)
{
struct sockaddr_in si_me, si_other;
int s, i, slen = sizeof(si_other) , recv_len;
char buf[BUFLEN];
//create a UDP socket
if ((s=socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == -1)
{
die("socket");
}
// zero out the structure
memset((char *) &si_me, 0, sizeof(si_me));
si_me.sin_family = AF_INET;
si_me.sin_port = htons(PORT);
si_me.sin_addr.s_addr = htonl(INADDR_ANY);
//bind socket to port
if( bind(s , (struct sockaddr*)&si_me, sizeof(si_me) ) == -1)
{
die("bind");
}
//keep listening for data
while(1)
{
printf("Waiting for data...");
fflush(stdout);
//try to receive some data, this is a blocking call
if ((recv_len = recvfrom(s, buf, BUFLEN, 0, (struct sockaddr *)
&si_other, &slen)) == -1)
{
```

```
die("recvfrom()");
}
//print details of the client/peer and the data received
printf("Received packet from %s:%d\n", inet_ntoa(si_other.sin_addr),
ntohs(si_other.sin_port));
printf("Data: %s\n" , buf);
//now reply the client with the same data
if (sendto(s, buf, recv_len, 0, (struct sockaddr*) &si_other, slen)
== -1)
{
die("sendto()");
}
}
close(s);
return 0;
}
```

Run the above code by doing a gcc server.c && ./a.out at the terminal. Then it will show waiting for data like this

```
$ gcc server.c && ./a.out
Waiting for data...
```

Next step would be to connect to this server using a client. We shall be making a client program a little later but first for testing this code we can use netcat.

Open another terminal and connect to this udp server using netcat and then send some data. The same data will be send back by the server. Over here we are using the ncat command from the nmap package.

```
$ ncat -vv localhost 8888 -u
Ncat: Version 5.21 ( http://nmap.org/ncat ) Ncat: Connected to 127.0.0.1:8888.
```

hello hello world world

Note: We had to use netcat because the ordinary telnet command does not support udp protocol. The -u option of netcat specifies udp protocol.

The netstat command can be used to check if the udp port is open or not.

```
$ netstat -u -a
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address Foreign Address State
udp 0 0 localhost:11211 *:*
udp 0 0 localhost:domain *:*
udp 0 0 localhost:45286 localhost:8888
ESTABLISHED
udp 0 0 *:33320 *:*
udp 0 0 *:ipp *:*
udp 0 0 *:8888 *:*
udp 0 0 *:17500 *:*
udp 0 0 *:mdns *:*
udp 0 0 localhost:54747 localhost:54747
ESTABLISHED
udp6 0 0 [::]:60439 [::]:*
udp6 0 0 [::]:mdns [::]:*
```

Note the *:8888 entry of output. Thats our server program.
The entry that has localhost:8888 in "Foreign Address" column, indicates some client connected to it, which is netcat over here.

Client

Now that we have tested our server with netcat, its time to make a client and use it instead of netcat. The program flow is like

socket() -> sendto()/recvfrom()

Here is a quick example

```
/*
Simple udp client
*/
#include<stdio.h> //printf
#include<string.h> //memset
#include<stdlib.h> //exit(0);
#include<arpa/inet.h>
#include<sys/socket.h>
#define SERVER "127.0.0.1"
#define BUFLEN 512 //Max length of buffer
#define PORT 8888 //The port on which to send data
void die(char *s)
{
perror(s);
exit(1);
}
int main(void)
{
struct sockaddr_in si_other;
int s, i, slen=sizeof(si_other);

char buf[BUFLEN];
char message[BUFLEN];
if ( (s=socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == -1)
{
die("socket");
}
memset((char *) &si_other, 0, sizeof(si_other));
si_other.sin_family = AF_INET;
si_other.sin_port = htons(PORT);
if (inet_aton(SERVER , &si_other.sin_addr) == 0)
{
```

```
fprintf(stderr, "inet_aton() failed\n");
exit(1);
}
while(1)
{
printf("Enter message : ");
gets(message);
//send the message
if (sendto(s, message, strlen(message) , 0 , (struct sockaddr *)
&si_other, slen)==-1)
{
die("sendto()");
}

//receive a reply and print it
//clear the buffer by filling null, it might have previously
received data
memset(buf,'\0', BUFLEN);
//try to receive some data, this is a blocking call
if (recvfrom(s, buf, BUFLEN, 0, (struct sockaddr *) &si_other, &slen)
{
die("recvfrom()");
}
puts(buf);
}
close(s);
return 0;
}
== -1)
{
die("recvfrom()");
}
puts(buf);
```

Plaintext = C^d mod n

Returning again to our numerical example, the ciphertext C = 82 would get decrypted to number 10 using private key 29 −

Plaintext = 82^{29} mod 91 = 10

RSA Implementation in Python

```
def rsa_algo(p: int,q: int, msg: str):
    # n = pq
    n = p * q
    # z = (p-1)(q-1)
    z = (p-1)*(q-1)
    # e -> gcd(e,z)==1      ; 1 < e < z
    # d -> ed = 1(mod z)        ; 1 < d < z
    e = find_e(z)
    d = find_d(e, z)

    # Convert Plain Text -> Cyper Text
    # Convert Plain Text -> Cypher Text
    cypher_text = ''
    # C = (P ^ e) % n
    for ch in msg:
        # convert the Character to ascii (ord)
        ch = ord(ch)
        # encrypt the char and add to cypher text
        # convert the calculated value to Characters(chr)
        cypher_text += chr((ch ** e) % n)

    # Convert Plain Text -> Cyper Text
    # Convert Plain Text -> Cypher Text
    plain_text = ''
    # P = (C ^ d) % n
    for ch in cypher_text:
```

```
        # convert it to ascii
        ch = ord(ch)
        # decrypt the char and add to plain text
        # convert the calculated value to Characters(chr)
        plain_text += chr((ch ** d) % n)
    return cypher_text, plain_text
def find_e(z: int):
    # e -> gcd(e,z)==1      ; 1 < e < z
    e = 2
    while e < z:
        # check if this is the required `e` value
        if gcd(e, z)==1:
            return e
        # else : increment and continue
        e += 1
def find_d(e: int, z: int):
    # d -> ed = 1(mod z)        ; 1 < d < z
    d = 2
    while d < z:
        # check if this is the required `d` value
        if ((d*e) % z)==1:
            return d
        # else : increment and continue
        d += 1
def gcd(x: int, y: int):
    # GCD by Euclidean method
    small,large = (x,y) if x<y else (y,x)
    while small != 0:
        temp = large % small
        large = small
        small = temp
    return large
if __name__ == "__main__":
```

Implement a client and a server on different computers using python. Perform the authentication of sender between these two entities by using RSA digital signature cryptosystem.

Cryptography Digital signatures

Digital signatures are the public-key primitives of message authentication. In the physical world, it is common to use handwritten signatures on handwritten or typed messages. They are used to bind signatory to the message. Similarly, a digital signature is a technique that binds a person/entity to the digital data. This binding can be independently verified by receiver as well as any third party.

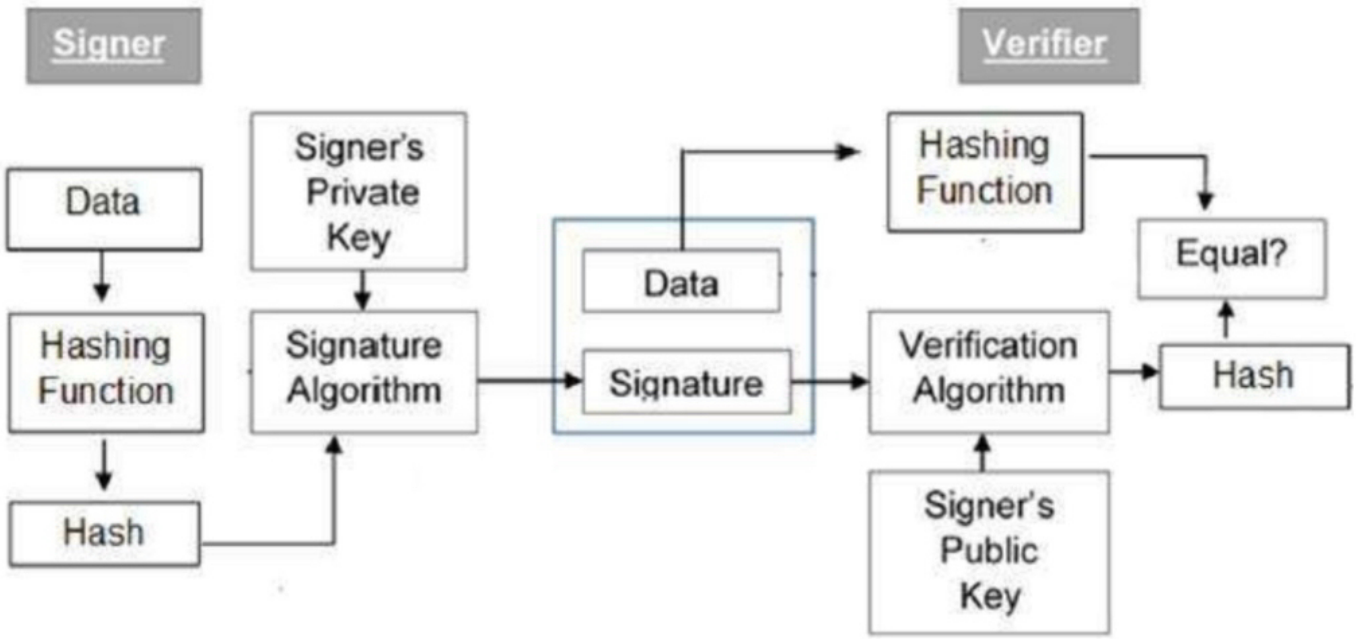
Digital signature is a cryptographic value that is calculated from the data and a secret key known only by the signer.

In real world, the receiver of message needs assurance that the message belongs to the sender and he should not be able to repudiate the origination of that message.

This requirement is very crucial in business applications, since likelihood of a dispute over exchanged data is very high.

Model of Digital Signature

As mentioned earlier, the digital signature scheme is based on public key cryptography. The model of digital signature scheme is depicted in the following illustration –



```
# Function to find gcd
# of two numbers
def euclid(m, n):
    if n == 0:
        return m
    else:
        r = m % n
        return euclid(n, r)


# Program to find
# Multiplicative inverse
def exteuclid(a, b):
    r1 = a
    r2 = b
    s1 = int(1)
    s2 = int(0)
    t1 = int(0)
    t2 = int(1)

    while r2 > 0:
        q = r1 // r2
        r = r1 - q * r2
        r1 = r2
        r2 = r
        s = s1 - q * s2
        s1 = s2
        s2 = s
        t = t1 - q * t2
        t1 = t2
        t2 = t

    if t1 < 0:
        t1 = t1 % a

    return (r1, t1)


# Enter two large prime
# numbers p and q
p = 823
q = 953
```

```
n = p * q
Pn = (p - 1) * (q - 1)

# Generate encryption key
# in range 1<e<Pn
key = []

for i in range(2, Pn):

    gcd = euclid(Pn, i)

    if gcd == 1:
        key.append(i)

# Select an encryption key
# from the above list
e = int(313)

# Obtain inverse of
# encryption key in Z_Pn
r, d = exteuclid(Pn, e)
if r == 1:
    d = int(d)
    print("decryption key is: ", d)

else:
    print("Multiplicative inverse for\
          the given encryption key does not \
          exist. Choose a different encryption key ")

# Enter the message to be sent
M = 19070

# Signature is created by Alice
S = (M ** d) % n

# Alice sends M and S both to Bob
# Bob generates message M1 using the
# signature S, Alice's public key e
# and product n.
M1 = (S ** e) % n
```

```
# If M = M1 only then Bob accepts
# the message sent by Alice.

if M == M1:
   print("As M = M1, Accept the\
        message sent by Alice")
else:
   print("As M not equal to M1,\
        Do not accept the message\
        sent by Alice ")
```

output

```
C:\Users\Nilofar\PycharmProjects\RSAProject\venv\Scripts\python.exe
C:/Users/Nilofar/PycharmProjects/RSAProject/rsadigital.py
decryption key is:  160009
As M = M1, Accept the   message sent by Alice
```

Python program for Diffie helmen Key exchange

```
from random import randint

if __name__ == '__main__':
    # Both the persons will be agreed upon the
    # public keys G and P
    # A prime number P is taken
    P = 23

    # A primitive root for P, G is taken
    G = 9

    print('The Value of P is :%d' % (P))
    print('The Value of G is :%d' % (G))

    # Alice will choose the private key a
    a = 4
    print('The Private Key a for Alice is :%d' % (a))

    # gets the generated key
    x = int(pow(G, a, P))

    # Bob will choose the private key b
    b = 3
    print('The Private Key b for Bob is :%d' % (b))

    # gets the generated key
    y = int(pow(G, b, P))

    # Secret key for Alice
    ka = int(pow(y, a, P))
```

```
# Secret key for Bob
kb = int(pow(x, b, P))

print('Secret key for the Alice is : %d' % (ka))
print('Secret Key for the Bob is : %d' % (kb))
```

Output

C:\Users\Nilofar\PycharmProjects\RSAProject\venv\Scripts\python.exe

C:/Users/Nilofar/PycharmProjects/RSAProject/diffehelmen.py

The Value of P is :23

The Value of G is :9

The Private Key a for Alice is :4

The Private Key b for Bob is :3

Secret key for the Alice is : 9

Secret Key for the Bob is : 9