Last Revision: April 17 2020. Original Version: July 4, 2019

So far I have installed WireGuard "servers" on a few single-board computers including the Raspberry Pi 1 Model B and the Raspberry Pi 3 Model B. I have also installed WireGuard "clients" on a couple of Android tablets, a recent low-end Android phone and on an old portable computer running Linux Mint 19. After a few false starts, I must admit that installation is not difficult especially as there is good information available on the Web. In this post I will give details about installing WireGuard on any Raspberry Pi 2 (version 1.2) and above running Raspbian Buster.

Table of Contents

1. Prerequisites to Installing WireGuard on a Raspberry Pi 2 v1.2 or above

I will describe how to install the WireGuard virtual network server on a Raspberry Pi 3 model B running Raspbian Buster Lite. The first installation was done on the June 2019 version of Buster but the last installation was done on the February 2020 version. There is no significant change between these two versions with regard to the installation of WireGuard as far as I can ascertain.

```
Raspbian Buster Lite
  Minimal image based on Debian Buster

  Version:        June 2019
  Release date:   2019-06-20
  Kernel version: 4.19

  Version:        February 2020
  Release date:   2020-02-13
  Kernel version: 4.19
```

The latest version of Raspbian is always available from the Raspberry Pi Foundation Raspbian downloads page. Full versions of Raspbian Buster are also available if that is the preferred OS. Even if the GUI version is installed, it will be necessary to open a terminal to install WireGuard.

If a newer version of Raspbian is used, then WireGuard may already be installed. As promised, WireGuard is now included in the Linux kernel. Indeed, wg was included in Armbian Bionic loaded onto an Orange Pi PC 2 a couple of months ago. Try the following commands.

```
pi@raspberrypi:~$ which wg
/usr/bin/wg
pi@raspberrypi:~$ which wg-quick
/usr/bin/wg-quick
```

If the two programs are found, WireGuard is installed so skip the rest of this section and section 3 and move on to Accessing the Local Network and the sections thereafter on configuring WireGuard. Otherwise the following steps must be performed beginning with a system update and the installation of the Linux kernel headers.

```
pi@raspberrypi:~ $ sudo apt update && sudo apt upgrade -y
...
31 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
Need to get 81.0 MB of archives.
After this operation, 5,875 kB of additional disk space will be used.
   Number of packages upgraded and the additional disk space used will depend on
   the last time the system was upgraded
...
```

```
pi@raspberrypi:~ $ sudo apt-get install raspberrypi-kernel-headers
Reading package lists... Done
...
The following NEW packages will be installed:
  raspberrypi-kernel-headers
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 16.7 MB of archives.
After this operation, 109 MB of additional disk space will be used.
...
Setting up raspberrypi-kernel-headers (1.20190517-1) ...
```

This process is lengthy. Note that in the latest version (Feb. 2020) of `Buster` it is version 1.20200212-1 of the kernel headers that is installed and that it is considerably bigger package.

That is it for the prerequisites in **Raspbian** `Buster`. The `dirmngr` utility, which performs network operations when managing and downloading certificates from Debian repositories is already installed in `Buster`.

2. Installing WireGuard in Raspbian `Buster`

Wireguard can be installed in `Buster` following the updated instructions from Adrian Mihalko and Ryan Govostes (Feb 10, 2020).

```
pi@raspberrypi:~ $ echo "deb http://deb.debian.org/debian/ unstable main" | sudo tee --append /etc/apt/sources.list.d/unstable.list
deb http://deb.debian.org/debian/ unstable main

pi@raspberrypi:~ $ wget -O - https://ftp-master.debian.org/keys/archive-key-$(lsb_release -sr).asc | sudo apt-key add -
--2020-04-18 19:56:18--  https://ftp-master.debian.org/keys/archive-key-10.asc
Resolving ftp-master.debian.org (ftp-master.debian.org)... 138.16.160.17
...
OK
pi@raspberrypi:~ $ printf 'Package: *\nPin: release a=unstable\nPin-Priority: 150\n' | sudo tee --append /etc/apt/preferences.d/limit-unstable
Package: *
Pin: release a=unstable
Pin-Priority: 150
pi@raspberrypi:~ $ sudo apt update
Get:1 http://deb.debian.org/debian unstable InRelease [146 kB]
...
All packages are up to date.
pi@raspberrypi:~ $ sudo apt install wireguard -y
Reading package lists... Done
...
Setting up wireguard (1.0.20200319-1) ...
Processing triggers for man-db (2.8.5-2) ...
pi@raspberrypi:~ $ which wg        checking
/usr/bin/wg
pi@raspberrypi:~ $ which wg-quick  checking
/usr/bin/wg-quick
```

I believe these instructions should also work if for some reason, installation is tried on **Raspbian Strech** as long as the utility `dirmngr` is in place before getting **WireGuard**. Note that I cannot confirm this. If you run into problems, you could consult an older post on the subject: Installing WireGuard on Raspbian Stretch and Buster.

3. Enabling Access to the Local Network

The whole point of hosting a virtual private network on a home network is to get secure access to that local network from anywhere on the Internet. Hopefully, that is it is not currently easily done because it would be a very bad idea to have the local network vulnerable to attacks from any bored *script kiddie* out there in the nasty world.

If you already have access to an IP camera, a home automation system or a self hosted cloud or NAS then you are probably quite familiar with dynamic host names and port forwarding so that you can skim through the next two steps, but do read carefully about IP forwarding.

3.1 Public IP Address and Dynanic DNS Host Name

All the computers and devices connected to your home network (often called LAN for Local Area Network) are assigned an IP address by the DHCP server which is probably the router provided by your Internet Service Provider (ISP). On my system the router has `192.168.1.1` for an IP address and the Raspberry Pi hosting the VPN server has a fixed IP address: `192.168.1.22`. If you think about it, there are many thousands of devices spread around the globe with that particular address. There is no hope that my Raspberry Pi can be reached from outside the LAN using `192.168.1.22` as the destination address. That problem has been solved with clever routing algorithms. All my devices connected to the local network that communicate with sites on the Internet send their traffic to the router at `192.168.1.1`. The router then passes each packet on to the ISP, changing the source IP address from say `192.168.1.22` to a public IP address assigned to my network by the ISP. That assigned public IP is unique on the whole of the Internet so that sites that receive packets from devices on my LAN, can reliably reply using as the destination IP the public IP address assigned by my ISP. When the router receives these packets of data, it routes them to the appropriate device on the LAN. There's obviously a little bit of magic going on to keep track of which device gets which packets as they come in, but that's another story. The point is that to talk to my Raspberry Pi from outside the LAN, the public IP address assigned by the ISP must be known. That's not difficult to find. Usually the router admin page shows that information and there are plenty of sites on the Web, such as What Is My IP Address and my-ip.com that will show you your public IP address.

If only it were that simple. Unfortunately, the public IP address cannot be trusted because it is dynamically assigned by the ISP and may change from time to time. Any DHCP server can force a client to reconnect at any time and change the assigned IP at that point. Also, when one logs off a network, the DHCP server will reserve the assigned IP for a certain "lease" time should the client connect again. After the lease time is expired, the IP address is returned to the pool of available addresses that the DHCP server

can assign to any new client. I have no idea just how long lease time is but it is not very short. Furthermore, devices like smart speakers and phones seem to be calling the mothership often enough to restart the lease so that I sometimes have the same public IP address for days on end. For the duration of this post, let's say that my sticky dynamic public IP address is `172.158.45.159`. Indeed, I could get away with using `172.158.45.159` as the public IP address of my network for testing the WireGuard configuration later on.

In practice though, one should avoid using a dynamic IP address. Instead the local network should be reached through a dynamic host name. DNS is the domain name system which translates the name of a website such as `www.google.com` into an IP address (`172.217.6.4`). Some sites offer a service, often free, that associates a domain name with an IP address. These sites update the IP addresses in their database at regular intervals. So, we will put in the HTML request the domain name obtained from the DNS service. The DNS will translate this name into an IP address that will be updated each time the ISP assigns a different IP address to the home server. There are plenty of sites on the Web that describe how to set up a dynamic domain name with any one of a number of DDNS providers and among them there is a description of how I did it using freedns.afraid.org back in 2018. So get yourself a dynamic host name, and learn how to signal any change in the public IP address assigned to your network to the DDNS service.

In what follows, my dynamic host name is deemed to be `modomo.twilightparadox.com`, which I hope is a fictitious name.

3.2 Port Forwarding

Part of the magic behind the routing of data packets across the router is that each packet must be sent through a "port". Ports are not physical entities, they are more like an apartment number added to a street address to ensure that a letter gets to the proper mail box. In the jargon, they are "end points" of a communication link and must be tacked on at the end of an IP address or host name. As an example, FTP control packets sent from the desktop computer to the Raspberry Pi, have as a destination address `192.168.1.22:21`. Some port numbers are implicit. All HTTP traffic is usually sent to port 80, while HTTPS traffic is sent to port 443. Try `https://www.google.com:9090` in a browser. This will send the request to port 9090, which is specified after the colon. The search engine does not "listen" to that port, so nothing will be displayed unless you are very patient and then some sort of error message may appear. Try `https://www.google.com:443` and you will see the familiar search page very quickly, but as you don't have to write the port number, it is implicit the HTTPS protocol.

For security reasons, consumer class routers such as the one supplied by an ISP have a built-in firewall that controls incoming and outgoing network traffic. Typically, outgoing traffic can only be sent out if the end point (i.e. port) is for some "well known" use. Typically, incoming traffic is blocked outright unless it is part of an exchange initiated by a device on the LAN. That is why you can use a Web browser from your home computer to read this post! There's an obvious problem for us. How can the Raspberry Pi be reached if the firewall will not let through IP packets destined to the Pi. So a "hole" has to be punched through the firewall. In technical terms, a port forwarding rule has to be established. That rule will instruct the firewall to send any IP packet addressed to the correct port to be sent on to a Pi.

OpenVPN which is a very popular VPN package uses a default destination port, 1194 to be precise, although that can be changed. Furthermore, whichever port OpenVPN uses, it will identify itself when queried with a port scanner. WireGuard does not have a default port nor will it reply if the port it does use is probed. That means that when configuring WireGuard later on, you will have to choose a port number. The latter are 16 bit integers, which means they have a range from 0 to 65435. However, choosing a number between 0 and 1023 is generally a bad idea. Typically, tutorials on the installation of WireGuard use relatively big numbers such as 53133 which are in the dynamic, private or ephemeral range.

It is difficult to give instructions about implementing port forwarding because each router model is different. On mine, there is a `Port Forwarding` tab in the `Basic` menu, and a `Add Rule` button which displays the window shown below when clicked.

As can be seen the router wants to forward a range of ports, so I specified a range of one port. I used the same port number for the public (Internet facing) port and for the private (local network) port. The latter will be appended to the local IP address, `192.168.1.22`. In my case, all IP traffic sent to `modomo.twilightparadox.com:53133` will end up at the outward facing edge of my router as traffic sent to `172.158.45.159:53133`. The router will then pass it over to the local network as traffic destined to being sent to `192.168.1.22:53133`. If everything is set up correctly, WireGuard will know what to do with it.

If you are having trouble setting up the port forwarding rules on your router, there are sites such as PF Network Utilities that have information about many router models. They also offer utilities that perform various including port forwarding, which I cannot endorse because I am much too paranoid to install such software and much too cheap to pay for it in the first place. I must say that the site provided accurate information about my router, but hidden it was hidden behind a lot of advertising for their products.

3.3 Enable IP Forwarding

If access to other LAN resources such as an IP camera or a Web server is needed, then IP forwarding has to be enabled on the computer hosting the WireGuard server. If you do not enable IP forwarding, you will not be taking full advantage of the virtual private network. I repeat, skipping IP forwarding only makes sense if the only device that needs to be reached from outside with the VPN is the WireGuard host machine.

```
pi@raspberrypi:~ $ cd /etc
pi@raspberrypi:/etc $ ls -l sysctl*
-rw-r--r-- 1 root root 2683 Apr  8 06:56 sysctl.conf

sysctl.d:
total 8
-rw-r--r-- 1 root root  51 Nov 26  2018 98-rpi.conf
lrwxrwxrwx 1 root root  14 Apr  8 07:51 99-sysctl.conf -> ../sysctl.conf
-rw-r--r-- 1 root root 639 May 17  2018 README.sysctl
```

Note how `/etc/sysctl.d/99-sysctl.conf` is a symbolic link to `/etc/sysctl.conf`. It will suffice to edit the later to enable IP packet forwarding.

```
pi@raspberrypi:/etc $ sudo nano sysctl.conf
```

Change

```
...
# Uncomment the next line to enable packet forwarding for IPv4
#net.ipv4.ip_forward=1
...
```

to

```
# Uncomment the next line to enable packet forwarding for IPv4
net.ipv4.ip_forward=1
```

as instructed in the configuration.file. A reboot will be necessary for the change to take effect.

```
pi@raspberrypi:~ $ sudo reboot
Connection to raspberrypi.local closed by remote host.
Connection to raspberrypi.local closed.
...
michel@hp:~$ ssh pi@raspberrypi.local
...
...
pi@raspberrypi:~ $ sysctl net.ipv4.ip_forward
net.ipv4.ip_forward = 1
```

4. Configuring WireGuard

The following list of steps might look daunting; it is actually rather easy to configure a **WireGuard** server and to add clients or peers.

4.1 Install the Adrian Mihalko User Management Script

One could follow Adrian Mihalko's guide to manually configure **WireGuard**, but I found his User Management Script very useful as I use **Android** tablets as clients on a regular basis. So what follows is mostly a copy and paste operation from the **GitHub** with just a few hints that might be useful for some.

```
pi@raspberrypi:~ $ sudo apt-get install git qrencode -y
...
0 upgraded, 6 newly installed, 0 to remove and 0 not upgraded.
Need to get 5,069 kB of archives.
After this operation, 26.8 MB of additional disk space will be used.
...
Setting up git (1:2.11.0-3+deb9u4) ...
Setting up qrencode (3.4.4-1) ...
pi@raspberrypi:~ $ git clone https://github.com/adrianmihalko/wg_config.git
Cloning into 'wg_config'...
remote: Enumerating objects: 50, done.
remote: Total 50 (delta 0), reused 0 (delta 0), pack-reused 50
Unpacking objects: 100% (50/50), done.
```

4.2 Generate the Private and Public Server Keys

```
pi@raspberrypi:~ $ cd wg_config
pi@raspberrypi:~/wg_config $ wg genkey | tee server_private.key | wg pubkey > server_public.key
   If you get a segmentation error here, then the Raspberry Pi was probably
     too old. It will be necessary to compile WireGuard.
pi@raspberrypi:~/wg_config $ cat server_public.key
5lFoBBjeLcJWC9xqS/Kj9HVwd0tRUBX/EQWW2ZglbDs=
pi@raspberrypi:~/wg_config $ cat server_private.key
aA+iKGr4y/j604LtNT+MQJ76Pvz5Q5E+qQBLW40wXnY=
   These two keys are needed for the next step. Copy them into
  a text editor on the desktop to easily copy them later.
```

4.3 Create and Edit the Server Definition File

```
pi@raspberrypi:~/wg_config $ cp wg.def.sample wg.def
pi@raspberrypi:~/wg_config $ nano wg.def
```

```
_INTERFACE=wg0
_VPN_NET=192.168.99.0/24
_SERVER_PORT=53133
_SERVER_LISTEN=wg.example.com:$_SERVER_PORT
_SERVER_PUBLIC_KEY=5lFoBBjeLcJWC9xqS/Kj9HVwd0tRUBX/EQWW2ZglbDs=
_SERVER_PRIVATE_KEY=aA+iKGr4y/j604LtNT+MQJ76Pvz5Q5E+qQBLW40wXnY=
```

If subnet `192.168.99.xxx` is used on the local area network, then the value of `_VPN_NET` will need to be changed. The `_SERVER_PORT` is the `UDP` port that will have to be forwarded to the WireGuard sever by the LAN router or gateway. This can be (perhaps should be) changed. **Change** `wg.example.com` in `_SERVER_LISTEN` to your dynamic domain name (I have a post about that Creating a dynamic domain name). In my (fictitious) case, that entry is:

```
_SERVER_LISTEN=modomo.twilightparadox.com:$_SERVER_PORT
```

Do not put the protocol prefix such as `https://`, just the domain name. As I explained above, the public IP address assigned to me by my ISP changes so rarely that I could get away with an IP address for testing purposes.

```
_SERVER_LISTEN=172.158.45.159:$_SERVER_PORT
```

Again, this is a fictitious value; sorry if it is your public IP.

4.4 Edit the Client Configuration Template

```
pi@raspberrypi:~/wg_config $ nano client.conf.tpl
```

```
[Interface]
Address = $_VPN_IP
PrivateKey = $_PRIVATE_KEY

[Peer]
PublicKey = $_SERVER_PUBLIC_KEY
AllowedIPs = 192.168.99.1/32, 192.168.1.0/24
Endpoint = $_SERVER_LISTEN
```

The value `$_PRIVATE_KEY` in the field `PrivateKey` will be changed to the **client** private key when a user (i.e. client) is generated. The first IP address in `AllowedIPs` is the IP address of the host of the WireGuard server being set up. The VPN network was set up as `_VPN_NET=192.168.99.0/24` in the `wg.def` file. So the server will be the first valid address, `192.168.99.1`, in that subnet. The second IP address, `192.168.1.0/24`, is actually a range: `192.168.1.0`, `192.168.1.1` and on up to `192.168.1.255`. It will be possible to reach all those addresses on the WireGuard server network from the client once the VPN tunnel is established. **Change** the second allowed IP to correspond to the real subnet used on your home local area network. It could be `192.168.1.xxx` as suggested above, `192.168.0.xxx` is often used, but some LANs use other private IPv4 addresses such as `10.0.3.xxx`.

4.5 Edit the Server Configuration Template

It is assumed that the name of the network interface used by the host of the WireGuard server is `eth0`. In my case that is not correct, the Raspberry Pi is connected by Wi-Fi to the LAN, so it was necessary to edit the sever configuration template.

```
pi@raspberrypi:~/wg_config $ nano server.conf.tpl
```

```
[Interface]
Address = $_SERVER_IP
ListenPort = $_SERVER_PORT
PrivateKey = $_SERVER_PRIVATE_KEY
PostUp = iptables -A FORWARD -i %i -j ACCEPT; iptables -A FORWARD -o %i -j ACCEPT; iptables -t nat -A POSTROUTING -o wlan0 -j MASQUERADE
PostDown = iptables -D FORWARD -i %i -j ACCEPT; iptables -D FORWARD -o %i -j ACCEPT; iptables -t nat -D POSTROUTING -o wlan0 -j MASQUERADE
```

The `eth0` interface is replaced by `wlan0` in the `PostUp` and `PostDown` lines that define changes to the server `iptables` when activating or deactivating the VPN.

4.6 Create an Empty WireGuard Server Configuration File

```
pi@raspberrypi:~/wg_config $ cd ..
pi@raspberrypi:~ $ sudo touch /etc/wireguard/wg0.conf
```

This is done once only. The user management script will update this file each time it is used to add or delete a user.

4.7 Enable Automatic Start of the wg0 Interface at Boot Time

```
pi@raspberrypi:~ $ sudo systemctl enable wg-quick@wg0
Created symlink /etc/systemd/system/multi-user.target.wants/wg-quick@wg0.service → /lib/systemd/system/wg-quick@.service.
```

This too is only done once, normally.

4.8 Start and Stop the WireGuard Interface Manually

```
pi@raspberrypi:~ $ sudo wg-quick up wg0
```

```
[#] ip link add wg0 type wireguard
[#] wg setconf wg0 /dev/fd/63
[#] ip link set mtu 1420 up dev wg0

pi@raspberrypi:~ $ sudo wg-quick down wg0
[#] ip link delete dev wg0
[#] iptables -D FORWARD -i wg0 -j ACCEPT; iptables -D FORWARD -o wg0 -j ACCEPT; iptables -t nat -D POSTROUTING -o eth0 -j MASQUERAD
```

4.9 Check on the Status of the Server

```
pi@raspberrypi:~ $ sudo systemctl status wg-quick@wg0
● wg-quick@wg0.service - WireGuard via wg-quick(8) for wg0
   Loaded: loaded (/lib/systemd/system/wg-quick@.service; enabled; vendor preset: enabled)
   Active: active (exited) since Fri 2019-03-05 16:50:49 ADT; 20min ago
     Docs: man:wg-quick(8)
           man:wg(8)
           https://www.wireguard.com/
           https://www.wireguard.com/quickstart/
           https://git.zx2c4.com/WireGuard/about/src/tools/man/wg-quick.8
           https://git.zx2c4.com/WireGuard/about/src/tools/man/wg.8
  Process: 378 ExecStart=/usr/bin/wg-quick up wg0 (code=exited, status=0/SUCCESS)
 Main PID: 378 (code=exited, status=0/SUCCESS)

Jul 03 16:48:58 domo systemd[1]: Starting WireGuard via wg-quick(8) for wg0...
Jul 03 16:50:46 domo wg-quick[378]: [#] ip link add wg0 type wireguard
Jul 03 16:50:46 domo wg-quick[378]: [#] wg setconf wg0 /dev/fd/63
Jul 03 16:50:47 domo wg-quick[378]: [#] ip address add 192.168.99.1/24 dev wg0
Jul 03 16:50:47 domo wg-quick[378]: [#] ip link set mtu 1420 up dev wg0
Jul 03 16:50:49 domo systemd[1]: Started WireGuard via wg-quick(8) for wg0.

pi@raspberrypi:~ $ sudo wg
interface: wg0
  listening port: 54130
```

There will be more information available when peers/clients have been set up. And even more information will be displayed when a client or peer has created a tunnel (i.e. opened a VPN).

5. Managing Users

Adrian Mihalko discusses setting up a mobile client on **IOS**. I will show how to set up a client on an **Android** table, an old Nexus 7, that I often bring along when I am outside the house. The first step is to install the **WireGuard** client application which is found in the **Google Play Store**. After that, here is the procedure used to configure the **WireGuard** server to accept a connection from the **Android** app.

```
pi@raspberrypi:~ $ cd wg_config
pi@raspberrypi:~/wg_config $ sudo ./user.sh -a nexus7
```

Don't worry about the QR code, it can be brought up later. In the meantime, here is the content of the user directory just created.

```
pi@raspberrypi:~/wg_config $ ls -l users/nexus7
total 24
-rw-r--r-- 1 root root 216 Jul  5 17:49 client.all.conf
-rw-r--r-- 1 root root 216 Jul  5 17:49 client.conf
-rw-r--r-- 1 root root 913 Jul  5 17:49 nexus7.all.png
-rw-r--r-- 1 root root 913 Jul  5 17:49 nexus7.png
-rw-r--r-- 1 root root  45 Jul  5 17:49 privatekey
-rw-r--r-- 1 root root  45 Jul  5 17:49 publickey
```

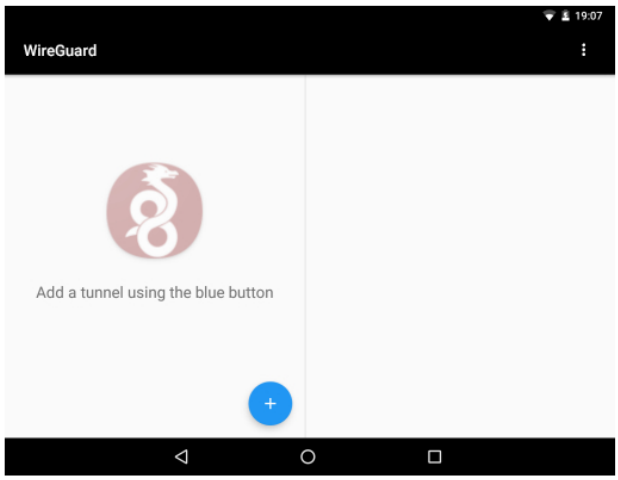
As with the **WireGuard** server, the client has two keys, private and public. Two client configuration files are created. One or both of these will be used to configure the **Android** client. It will be easy to do using the two png images which are QR codes containing the information about each of the client configuration files. Here is the content of the client configuration file and the updated server configuration file.

```
pi@raspberrypi:~/wg_config $ cat users/nexus7/client.conf
[Interface]
Address = 192.168.99.2/24
PrivateKey = gH5xInhP2NZw0t8hVgJPhTRDUh3Bir7FEynRcW8IHlg=

[Peer]
PublicKey = 5lFoBBjeLcJWC9xqS/Kj9HVwd0tRUBX/EQWW2ZglbDs=
AllowedIPs = 192.168.99.1/32, 192.168.1.0/24
Endpoint = wg.example.com:53133

pi@raspberrypi:~/wg_config $ cat users/nexus7/client.all.conf
[Interface]
Address = 192.168.99.2/24
PrivateKey = gH5xInhP2NZw0t8hVgJPhTRDUh3Bir7FEynRcW8IHlg=

[Peer]
PublicKey = 5lFoBBjeLcJWC9xqS/Kj9HVwd0tRUBX/EQWW2ZglbDs=
AllowedIPs = 0.0.0.0/0
Endpoint = wg.example.com:53133

pi@raspberrypi:~/wg_config $ sudo cat /etc/wireguard/wg0.conf
[Interface]
Address = 192.168.99.1/24
ListenPort = 53133
PrivateKey = aA+iKGr4y/j604LtNT+MQJ76Pvz5Q5E+qQBLW40wXnY=
PostUp = iptables -A FORWARD -i %i -j ACCEPT; iptables -A FORWARD -o %i -j ACCEPT; iptables -t nat -A POSTROUTING -o wlan0 -j MASQUERADE
PostDown = iptables -D FORWARD -i %i -j ACCEPT; iptables -D FORWARD -o %i -j ACCEPT; iptables -t nat -D POSTROUTING -o wlan0 -j MASQUERADE
[Peer]
PublicKey = BEnqBZ6rWcDO6lKhb6oXM7aRvE7fuIWCZw1PxgyMMyE=
AllowedIPs = 192.168.99.2/32
```

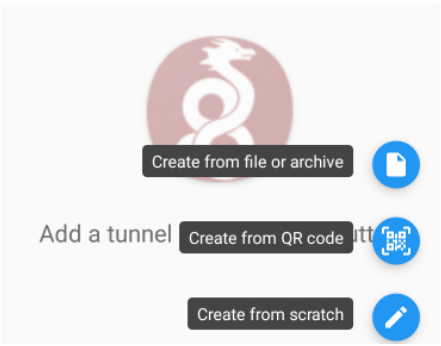
The two client configuration files are identical except for the `AllowdIPs` field. Display the user information again. This time the two configuration files and the two QR codes will be displayed, but it will be necessary to scroll back to see them.

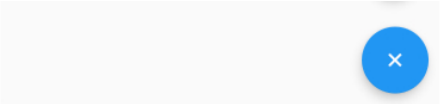
```
pi@raspberrypi:~/wg_config $ sudo ./user.sh -v nexus7
```

Now it is time to install the WireGuard Application from **Google Play** on the **Android** device. Launch the application.



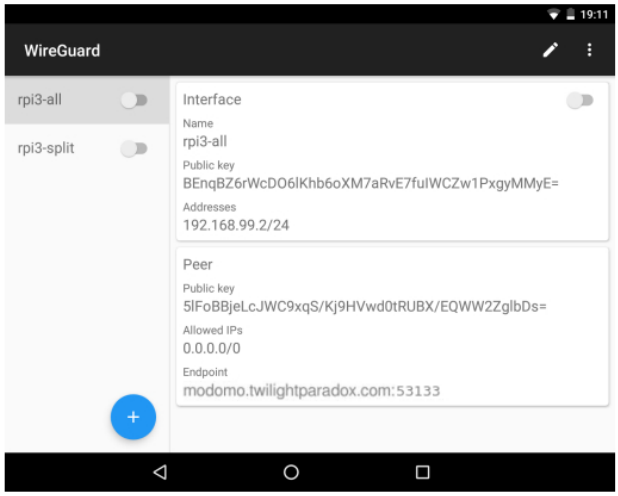
Click on the blue button as told.





On a newer Android phone, the choices are displayed in a different fashion, but the same three choices are offered. Click on **Create from QR code**. Aim the camera towards the QR code displayed on the desktop monitor. Once the information is acquired, the following dialog appears.

I named the tunnel "Rpi3-split" and then pressed on the CREATE TUNNEL button. I repeated the steps to add the second tunnel, named "RPi-all", from the second QR code.

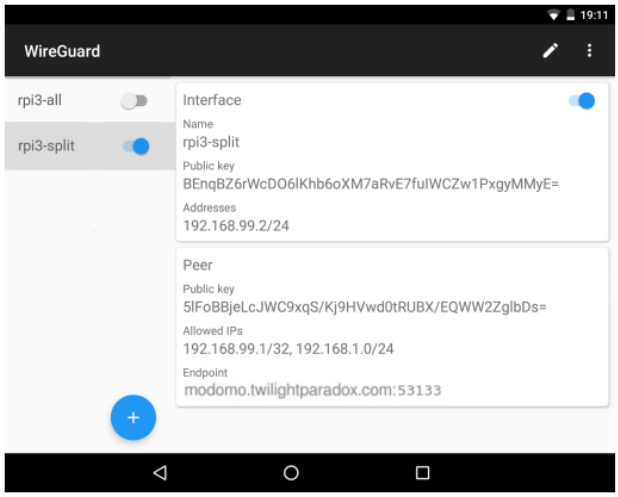


The above screen capture done on the older tablet shows the appearance of the **Android WireGuard** application once the two tunnels were created . When selecting a tunnel, the "public" information is displayed on the right panel. On the newer telephone, only the tunnel names appear on the screen. In other words, only the left half of the image shown above is visible. Click on a tunnel name to display its configuration information. Click on the left arrow at the top to come back to the list of tunnels.

I have not found a simple way to remove a server entry in the **WireGuard** app. What I do instead is to `Export tunnels to zip file` in the app menu (the three vertical dots ⋮). Then I copy the zip file over to my desktop machine over a USB connection with the tablet. The content of the zip file can be edited removing any unwanted server. The modified zip file is copied back to the tablet. I then erase all application settings. In the older version of **Android** installed on my tablet, this is done in `Settings, Storage an USB, Applications, WireGuard, Erase Data`. When the **WireGuard** app is started, all tunnels will be gone and pressing on the blue + button will bring up the menu already shown before. Instead of choosing **Create from QR Code** choose **Create from file or archive** to import the modified zip file containing the tunnel definitions.

6. Using the WireGuard VPN

Imagine the following scenario. I am sitting in a coffee shop, and I want to see the video feed from an IP camera at home. On the local network, I would start VLC and view the stream at the following address: `rtsp://192.168.1.95/11`. On my tablet, I can do exactly the same thing as long as I start the **WireGuard** application and open one of the tunnels to the VPN server at home.



I just slide the wanted tunnel button to the right as shown above. On my **Android** phone the connection details would not be displayed but opening a tunnel would be done just the same, by sliding to the right the control beside the desired tunnel. As soon as that is done, I have access to all resources on my home network on `192.168.1.xxx` just as if my tablet were connected directly to the LAN. I can therefore watch the `rtsp://192.168.1.95/11` video stream as if I were home. So simple and yet secure. Anyone eavesdropping on the Wi-Fi network in the shop or anywhere along the route between my tablet and my home router would see IP packets with encrypted content.

Instead of seeing the address `192.168.1.95:554` from which it could be surmised that there is an IP camera on my home network (554 is the typical RTSP port), the visible address will be `172.158.45.159:53133` which is the public IP address of the router and the obscure port used by the `WireGuard` interface which encodes everything else end-to-end, including the final destination address.

Note that the client gives very little feedback. A little key icon signifying the VPN is active will be shown at the top of the tablet screen. But that icon is present even if the settings are wrong or if the **WireGuard** server at home is not online. The only "symptom" that something is wrong will be that all devices on the `192.168.1.xxx` subnet are unreachable!

What if I decide to consult the latest stock market indices? I would start a web browser and go to the say **Yahoo! Finance** in Canada: `https://ca.finance.yahoo.com/`. The destination IP, `66.218.84.42`, is not on the `192.168.1.xxx` subnet so routing of the packets would not go through the **WireGuard** tunnel. Instead, packets will be routed directly as if **WireGuard** were not even running.

If I then want to check my bank balance, I can either start a Web browser and establish a secure HTTPS connection with the bank's Web server or use the **Google Play Store** app provided by the bank. Either way, I am counting on the built-in encryption of the data exchanged to keep my password and the details of my finances private. However, being paranoid, before checking the balance, I usually start the other tunnel that I named `rpi3-all` where the `Allowed IPs` field is `0.0.0.0/0`. That means all traffic in and out of my tablet in the coffee shop is sent to my home network and from there it is routed to its final destination. So my outgoing financial data is double encrypted on the first leg of its journey out of the coffee shop and incoming data is also double encrypted on the last leg. Maybe I should wear a tin foil hat to protect myself from the nefarious 5G network at the same time because for most of the way, the data is transiting all sorts of bridges, routers, backbones and so on with no more and no less encryption than when I consult my bank balance from my desktop computer at home. Still I find it reassuring to use the "universal" **WireGuard** tunnel at all times when using a public hotspot. I find there is no perceptible slow down even with the extra hop involved.

Once you have thoroughly tested everything, I suggest it is time to look at all ports that were being forwarded at the LAN firewall. I was able to remove all but one hole punched through for the home automation system, for IP cameras and so on. That was the impetus to use **Amazon Alexa** exclusively for voice activated home automation tasks and thus get rid of IFTTT with which was not very reliable. Now there's a single small black hole in the firewall. Try it and you too may get a warm fuzzy feeling of security. Hopefuly, I will not regret this in the future.

7. Installing WireGuard on Other Devices

On older Raspberry Pi models it is necessary to compile **WireGuard** from the source code. Again, I followed Adrian Mihalko detailed instructions to install the software correctly on a Raspberry Pi 1. I was surprised that the VPN performed adequately even when routing all Internet traffic through it in a coffee shop for better security. My bandwidth demands were usually relatively light. Nevertheless, YouTube videos could be streamed simultaneously on a tablet and portable without noticeable degradation. Of course these were not in high-definition, but then I have no need to view 4K videos in coffee shops in the forseeable future.

Not long after installing **WireGuard** on the older Raspberry Pi, I did the same on an Orange Pi Zero with an older version of **Armbian**. It was necessary to compile the source code much in the same way as on the older Raspberry Pi, but there was a bit of a struggle to get the prerequisite packages in place. There is no point in pursuing this any further given that the latest version of **Armbian** `Bionic` includes **WireGuard**.

The fact that **WireGuard** is now part of the **Linux** kernel will only hasten its adoption by an ever-growing number of users. In my opinion, that is a good thing as I have found that using it to host a VPN is very reliable, useful and surprisingly seamless. I suspect that soon installation will no longer be a barrier even on **Rasbian**, and there will be no real argument against at least giving it a try.
