WireGuard

Basics of WireGuard

What is WireGuard?

WireGuard is a fairly new Layer 3 VPN protocol that runs on top of UDP. It's main selling points are that it is very **easy** to configure and also very **fast** (It can easily reach speeds in excess of 1 Gbit/s without much resource utilization).

It is available for every major operating system you can find in the wild (namely Windows, Linux (and its derivatives) and Mac OS)

To understand how simple the configuration is, have a look at the following snippet:

```
[Interface]
Address = 10.0.0.1/24
ListenPort = 51820
PrivateKey = 4BhSWsplXEJrqSDvb/kIy6FzfXkimLF4b3h/nrz/vkY=

[Peer]
PublicKey = EzEyCarbQdia+D0u7aRvSDL4hz3YCQQjgvOVGBEPBDo=
AllowedIPs = 10.0.0.2/32

[Peer]
PublicKey = dq/DbVCb40ZjuKZZv1EhTH/4FQRGme4pe07B5CIvuVo=
AllowedIPs = 10.0.0.3/32

[Peer]
PublicKey = UDIqfBJAKc1YmpvRxDSM4tc3ZrbzNHduEqVqZmG4CXU=
AllowedIPs = 10.0.0.4/32
```

That small little amount of config would allow 3 clients to connect to the WireGuard "server" with their own matching private keys!

Want to use WireGuard for your own network? Great! See the rest of this documentation here!

How does WireGuard work?

WireGuard works by using cryptographic keys to encapsulate packets, then sending them using it's own routing table, It's generally regarded as a secure and fast way to send packets, and is highly scalable and deployable!

Generating a public/private keypair

Generating a public and private key in wireguard is super easy! You'll know how to do it in a matter of seconds :D

To generate a private key you should run this

```
wg genkey | sudo tee /path/to/where/you/want/your/private.key
```

This will generate your private key, print it to your terminal and write it to a file in the path you specify above

To get the matching public key for this private key, you need only run this!

```
sudo cat /wherever/your/private/key.is | wg pubkey | sudo tee /path/to/your/new/public.key
```

WireGuard configuration options

[Interface]

The [Interface] config option for the local device running the tunnel

Used as

```
[Interface]
```

Address

`Address`

This config option defines the addresses for the device being configured. It goes under the [Interface] config block and it varies depending on what the wireguard node is doing. If this node is acting as a router for other clients on the network (bounce node/router) It should be a subnet assigned to the interface, similar to how you would assign a subnet to a routers interface. IG; 192.0.2.1/24 or 2001:DB8::/64

Specifying multiple addresses should be done with comma separated values of each subnet/address using CIDR notation always.

IE; Address = 192.0.2.1/24,2001:DB8::/64

ListenPort

ListenPort is a config option for hard programming a port to bind to, this is typically used on routers/bounce nodes/relays that are relaying traffic for other devices. Wireguard is ***always*** UDP and cannot be set to use TCP.

Example usage is ListenPort = 1600
The default wireguard port is 51820

PrivateKey

The PrivateKey config option defines the private key for the node you are configuring. Not much else to say about this besides **Do not give your private key to anyone**

Example is PrivateKey = thisisaprivatekeybcdabcdabcdabcda=

You can learn how to generate a private key here!

DNS

The DNS config flag sets the DNS servers for use with the tunnel, and is generally set on devices redirecting all there traffic via a bounce node (typical commercial VPN style setup)

Example usage is
Dns = 1.1.1.1
For more than one, separate the values with commas, like
Dns = 1.1.1.1,1.0.0.1

Table

The Table flag is crucially important for use with more advanced setups, like passing BGP traffic and routes via a wireguard tunnel, there are multiple options, auto being the default, where wireguard makes it's own table and adds routes by itself, which is fine for clients, but bad if you want BGP to make routes, the second option is to specify a specific table you want routes added to. The third option is Table = off, which adds no routes automatically, and is ideal when you are using dynamic routing engines like BGP.

Examples are below

Auto

```
Table = auto (or don't specify this option, this is the default)
```

Specific table

```
Table = 2345 (or any table you want)
```

No tables

```
Table = off
```

MTU

The MTU flag as you would probably guess sets the MTU of the tunnel, it's default is 1420 or whatever your upstream internet line is set to. but using this command you can lower this (or raise it, but be warned unless your entire path supports jumbo frames this is likely to break your connection on a standard 1500 MTU line). Basically don't touch this unless you have a need to

Example usage is below

```
MTU = 1392
```

PreUp

PreUp is used to run a command before your tunnel interface is brought up. This command can also be used more than once, and is very useful for adding static routes for tunnels that accept full BGP tables and other specific use cases

Example is below

```
PreUp = ip ro add 1.1.1.1 dev eth0 via 4.4.3.1
```

PostUp

Similar to Preup PostUp is used to run commands, but PostUp runs after the tunnel is up and running. This is useful for adding firewall rules or internal to tunnel routes after it's been built. Example is below

```
PostUp = ip addr add 4.5.6.2/32 dev wg0
```

PreDown

The PreDown command is used to run commands on request the tunnel be destroyed/brought down, it's the inverse of PreUp and is useful to remove things added on the construction of the tunnel that depend on it existing, like IP addresses on it's interface

Example is below

```
PreDown = ip addr del 4.5.6.2/32 dev wg0
```

PostDown

The PostDown similar to the others is the inverse of PostUp and will run commands on the successful destruction of the tunnel. useful for removing things you added for the tunnel to work initially

Example below!

```
PostDown = ip ro del 1.1.1.1 dev eth0 via 4.4.3.2
```

[Peer]

[Peer] is used for defining the VPN settings for a remote node capable of routing traffic for one or more addresses being itself and other devices attached to it. Peers can be either a router style box that passes traffic to other peers, or a client via LAN/internet that is not behind a NAT and only routes traffic for itself (typically a client device like a phone or laptop).

Endpoint

Defines the public IP:port of a remote peer or device. Do not include this for devices that sit behind a NAT or otherwise do not have a static public IP address as it will break the connection when it changes. This should only be defined if the peer has a static, unchanging address, like a bounce/router node.

Example below

```
Endpoint = 5.5.5.5:5000
Endpoint = 2601:2601::1:5000
```

AllowedIPs

AllowedIPs defines any IP ranges for which the device will be routing or passing traffic for. Client devices like laptops and phones will generally only have one or two IPs, being an IPv4 and IPv6 address of the client. On router nodes/bounce boxes this should be the subnets for which the router/bounce node will handle. It can be used more than once in the config file. You can specify multiple by using commas to separate the values, or you can just add multiple lines of AllowedIps. Similar to other routing engines it will prefer shortest length paths first, so if you have 1.1.1.1/32, 1.1.1.0/24, and 0.0.0.0/0 it will always pick 1.1.1.1/32 first.

Examples below!

```
peer is a router for other peers
AllowedIPs = 192.0.2.1/24, 2601:2601::/48

peer is a relay server that routes to itself and all nodes on its local interface
AllowedIPs = 192.0.2.3/32,192.168.1.1/24

peer is a relay devices that routes to itself and only one other peer
AllowedIPs = 192.0.2.3/32,192.0.2.4/32

peer is a routing server that bounces all internet & VPN traffic (similar to commercial VPNs)
AllowedIPs = 0.0.0.0/0,::/0

peer is a client device that only does traffic for itself.
AllowedIPs = 192.0.2.3/32, 2601:2601:c::3/128
```

PublicKey

This config option is used to define the public key of the remote node! to find out how to get the public IP of a remote node you created, check out this page!

If you're using someone elses wireguard server, this should have been provided to you on setup.

Example of use below :D

```
PublicKey = thisisapublickey12341231234=
```

PersistentKeepalive

PersistentKeepalive is used for devices that are behind nats or other restrictive firewalls to prevent an idle connection from being terminated by the firewall, it works by periodically sending traffic over the tunnel to tell firewalls and NAT devices in the way to keep the session alive. Avoid using this unless you need it as it uses un-needed traffic otherwise.

Example below ^^

```
PersistentKeepalive = 25
(this is a sane value for most NATs, if it's particularly restrictive or quick to close, try lowering the value.)
```

WireGuard example configs

Simple router/bounce node with 3 clients

DO NOT USE THE KEYS IN THESE EXAMPLES, THEY ARE PUBLIC AND SHOULD NEVER BE USED IN PRODUCTION

Bounce node config

```
[Interface]
Address = 10.0.0.1/24
ListenPort = 51820
PrivateKey = sFHggcBxny9La+jQIQEZjrF4eT6U6IZ7kIbE9Xt+I14=

[Peer]
PublicKey = TruSQaN5wVCTgdWNkTin+h21JbJUXy6tNoSRixBMXlQ=
AllowedIPs = 10.0.0.2/32

[Peer]
PublicKey = ebkxqf5sUC41lZIX2sfdt0XHTdyr+Ii2Ox7CIqwcpHQ=
AllowedIPs = 10.0.0.3/32

[Peer]
PublicKey = DJ8OhWf4rOG/KsYrmrLwUheI7r+bv0oiMCcYXvjsnUQ=
AllowedIPs = 10.0.0.4/32
```

Client 1

```
[Interface]
Address = 10.0.0.2/24
ListenPort = 51820
PrivateKey = wO5JlyMuZBydG8r/VcmeYgag4A2+w9ChSu2tS9uTbkA=

[Peer]
PublicKey = EFEa6B4ZyajlVHBguXAhTjixiR7hr6PbhCn4EeA46m4=
AllowedIPs = 0.0.0.0/0, ::/0
Endpoint = myserver.dyndns.org:51820
```

Client 2

```
[Interface]
Address = 10.0.0.3/24
ListenPort = 51820
PrivateKey = uGcpY6TmDovligr0pB+rRUblExRcoH7ohFGFyOTJH38=

[Peer]
PublicKey = EFEa6B4ZyajlVHBguXAhTjixiR7hr6PbhCn4EeA46m4=
AllowedIPs = 0.0.0.0/0, ::/0
Endpoint = myserver.dyndns.org:51820
```

Client 3

```
[Interface]
Address = 10.0.0.4/24
ListenPort = 51820
PrivateKey = 4LsV6UgThgUUtGsJu9o1JapbhTy6p4Mbixb0YsJ8lmA=

[Peer]
PublicKey = EFEa6B4ZyajlVHBguXAhTjixiR7hr6PbhCn4EeA46m4=
AllowedIPs = 0.0.0.0/0, ::/0
Endpoint = myserver.dyndns.org:51820
```