Crypto routing with WireGuard: Proof-of-a-concept implementation

March 24, 2025

Abstract

WireGuard is a recent development of a novel VPN protocol that requires 1-RTT is used to establish a secure, authenticated channel between peers. WireGuard revolves around the idea of crypto routing - routing based on the public keys of the peers. WireGuard uses EC curve X25519 for identification and authentication, and a set of novel algorithms to secure data plane traffic - ChaCha20 and Blake2s algorithms. In this document, we present a simple implementation of WireGuard protocol in userspace using Python language. Our implementation, although is a proof-of-concept, achieves 70Mb/s throughput which is sufficient for single-user deployments.

1 Background

Wireguard [2] is a novel security protocol that allows for building secure Virtual Private Networks on top of layer 3. Wireguard uses the notion of a cryptorouting in which the forwarding of a packet occurs based on the public key of a peer. Wireguard is a 1-RTT protocol, meaning that it requires (in the best case) a single round-trip to negotiate the keys for securing the data plane traffic. In addition, Wireguard introduces the concept of cryptographic cookies which can be used during the heavy load time instants to protect the server from denial-of-service attacks. Finally, Wireguard uses the latest cryptographic algorithms such as Authenticated Encryption with Authentication Data (AEAD) ChaCha20 algorithm and compact 256-bit elliptic curve X25519. In what follows we will describe the setup of Wireguard between two nodes deployed on virtual machines.

2 Environment setup and basic results

In order to run the Wireguard in the test environment first one needs to checkout the Wireguard package from GitHub. Thus, on each peer perform the following command:

```
git clone https://github.com/dmitriykuptsov/wireguard.git
```

Then one needs to generate a pair of X25519 EC keys:

```
$ cd wireguard
$ python3 tools/genkey.py
```

Copy the keys and save then in config/configuration.txt file. Also in this file modify the peer address (address visible on the Internet or Local Area Network (LAN) address which will be used for port forwarding if you are behind the network address translator (NAT)), UDP port number, and local address of Tunneling interface (this will be used in Wireguard routing process).

Once the file is modified on both sides you can run the Wireguard daemon as follows:

```
$ sudo python3 wg.py
```

Bellow we assume that peer A has 10.1.1.5 as TUN interface address, while peer B has 10.1.1.6. We also assume that peer A's public key is $8aW8c...ZBkE=$ (of course this is a shorten version), while peer B has $mRpAN...KHG4=$. Run the following commands on peer A:

```
sudo ip route add 10.1.1.0/24 via 10.1.1.5
```

Then on peer B:

```
sudo ip route add 10.1.1.0/24 via 10.1.1.6
```

Add cryptographic routing entry, for that connect to WG daemon using *nc* utility as follows:

```
$ nc -vv localhost 10000
```

Then copy the routing entry (of course you should change the parameters that match your setting), and press Enter key:

```
add route 10.1.1.6 255.255.255.255 mRpAN...KHG4= 14501 192.168.64.19
```

Repeat the same procedure on the other side - peer B:

```
add route 10.1.1.0 255.255.255.0 8aW8c...ZBkE= 14500 192.168.64.16
```

Now go to peer A and test the connectivity between the nodes:

```
dmitriy@initiator:~$ ping 10.1.1.6
PING 10.1.1.6 (10.1.1.6) 56(84) bytes of data.
64 bytes from 10.1.1.6: icmp_seq=1 ttl=64 time=5.77 ms
64 bytes from 10.1.1.6: icmp_seq=2 ttl=64 time=5.16 ms
```

Once the secure tunnel is established you should see the ICMP packets flowing back and forth.

To understand the performance of our proof-of-concept implementation using Python language we have performed a series of throughput tests (with the help of the Iperf tool). All in all, we have reached the maximum of 70Mb/s between a pair of virtual machines hosted on a MacBook M1 laptop.

3 Conclusions

In this short document, we demonstrated Python-based implementation of the Wireguard protocol to the interested reader. We have tested the protocol using a pair of Ubuntu servers and measured the throughput. All in all, we have obtained a maximum of 70Mb/s between a pair of Wireguard peers, which we believe is sufficient for most use cases involving clients communicating over DSL, ADLS, 100Mb/s Ethernet, and the like. We have exposed our proof-of-a-concept implementation of the Wireguard protocol using Python language in our GitHub repository. Interested readers can find it here [1].

References

[1] WireGuard: Python implementation. https://github.com/dmitriykuptsov/wireguard, 2025.

[2] J. A. Donenfeld. *WireGuard: Next Generation Kernel Network Tunnel*. 2020.