WireGuard

A next generation VPN tunnel

PRESENTED BY QINGWEI “VINCENT” ZHANG

MARCH 7, 2019.

OTTAWA CANADA LINUX USER GROUP.

Who am I

- Qingwei Zhang, a software development engineer with 5 years of experience in high technology and finance.
- A previous small business entrepreneurs
- Background in computer networks
- Motivated to introduce a VPN that avoids the problems in both crypto and implementation

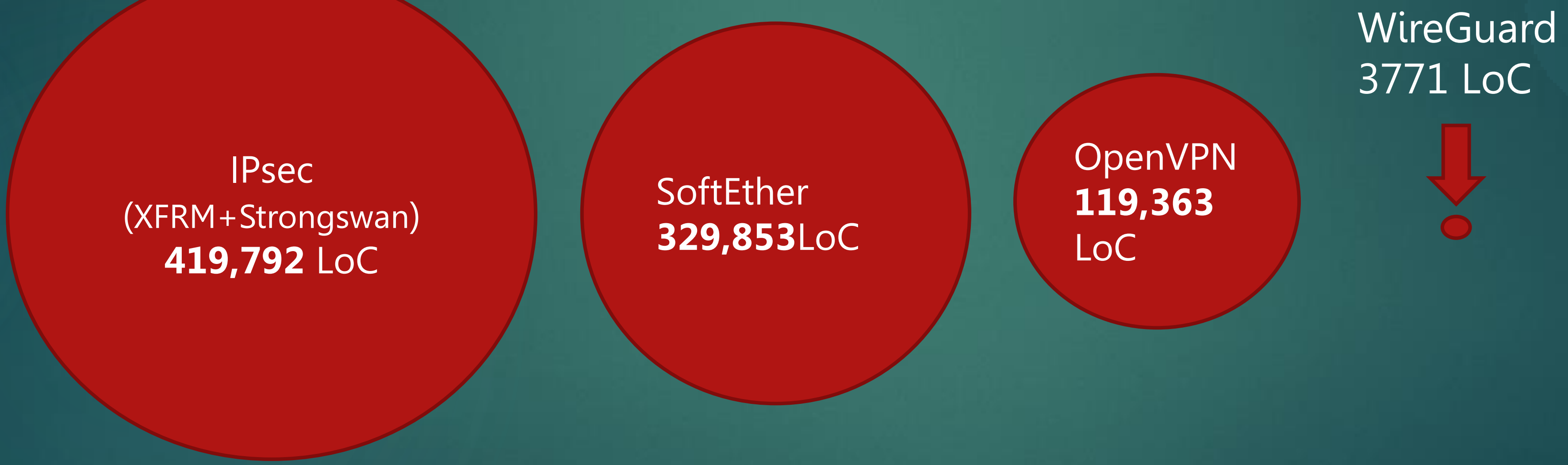
What is WireGuard?

- Layer 3 secure network tunnel for IPv4 and IPv6.
- *Designed for* the Linux kernel
  - Slower cross platform implementations.
- UDP-based. Punches through firewalls.
- Modern conservative cryptographic principles.
- Emphasis on simplicity and auditability.
- Authentication model similar to SSH's ./.ssh/authenticated_keys.
- Replacement for OpenVPN and IPsec.
- Grew out of a stealth rootkit project.

Security Design Principle 1: Easily Auditable

OpenVPN	Linux XFRM	StrongSwan	SoftEther	WireGuard
116,730 LoC Plus OpenSSL!	119,363 LoC Plus StrongSwan!	405,894 LoC Plus XFRM!	329,853 LoC	**3,771 LoC**

Security Design Principle 1: Easily Auditable



Security Design Principle 2: Simplicity of Interface

- WireGuard presents a normal network interface:

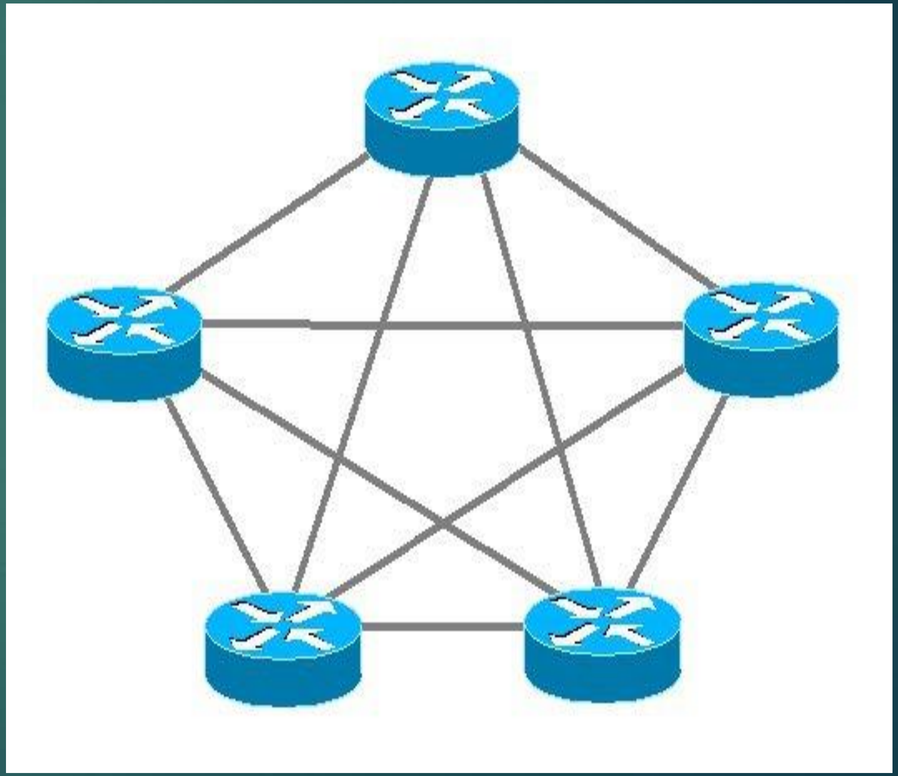
```
# iplink add wg0 type WireGuard
# ipaddress add 192.168.3.2/24 dev wg0
# iproute add default via wg0
# ifconfig wg0 …
# iptables–A INPUT -iwg0 …
```

```
/etc/hosts.{allow,deny}, bind(), …
```

- Everything that ordinarily builds on top of network interfaces –like eth0or wlan0–can build on top of wg0.

Cryptokey Routing

- **The fundamental concept of any VPN is an association between public keys of peers and the IP addresses that those peers are allowed to use.**
- A WireGuard interface has:
  - A private key
  - A listening UDP port
  - A list of peers
- A peer:
  - Is identified by its public key
  - Has a list of associated tunnel IPs
  - Optionally has an endpoint IP and port



Cryptokey Routing

PUBLIC KEY :: IP ADDRESS

CryptokeyRouting

Server Configure

[Interface]

PrivateKey= yAnz5TF+lXXJte14tji3zlMNq+hd2rYUIgJBgB3fBmk=

ListenPort= 41414

[Peer]

PublicKey= xTIBA5rboUvnH4htodjb6e697QjLERt1NAB4mZqp8Dg=

AllowedIPs= 10.192.122.3/32,10.192.124.1/24

[Peer]

PublicKey= TrMvSoP4jYQlY6RIzBgbssQqY3vxI2Pi+y71lOWWXX0=

AllowedIPs= 10.192.122.4/32,192.168.0.0/16

Client Configure

[Interface]

PrivateKey= gI6EdUSYvn8ugXOt8QQD6Yc+JyiZxIhp3GInSWRfWGE=

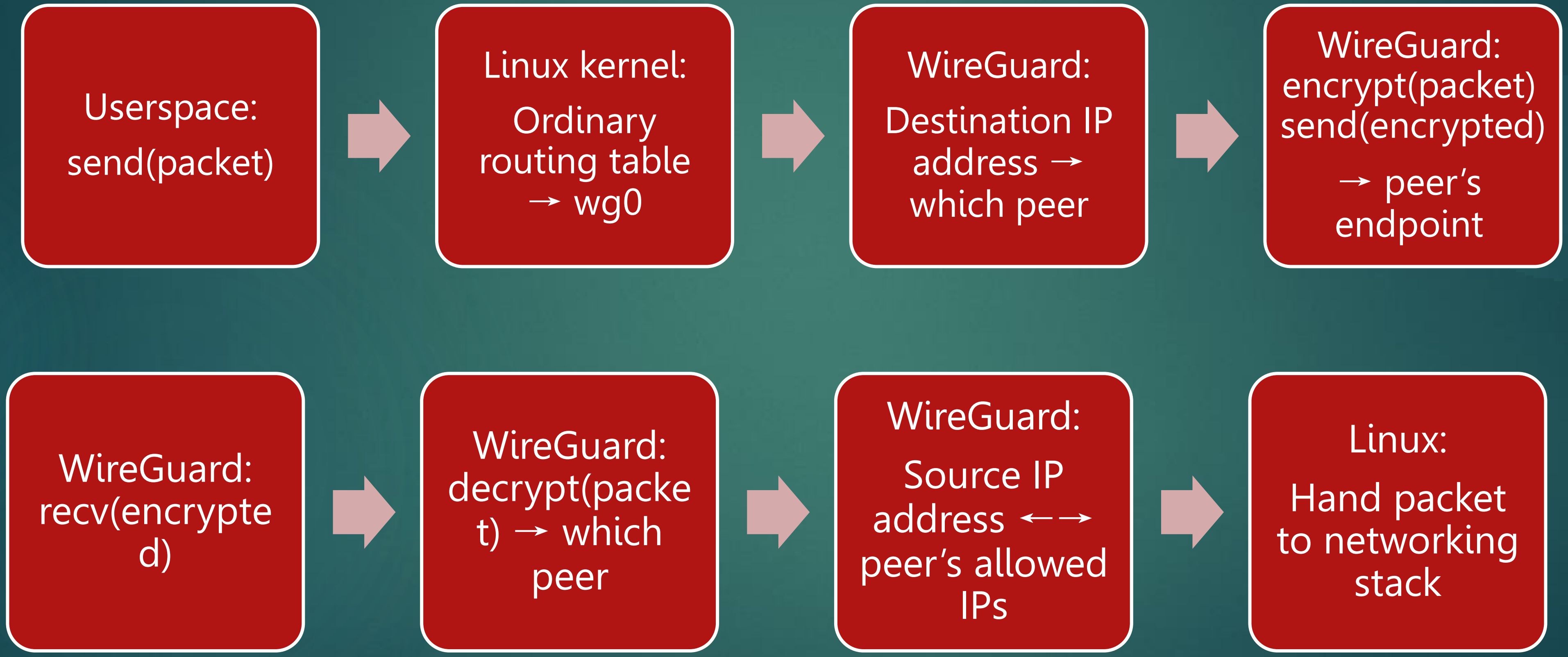
ListenPort= 21841

[Peer]

PublicKey=
HIgo9xNzJMWLKASShiTqIybxZ0U3wGLiUeJ1PKf8ykw=Endpoint =
192.95.5.69:41414

AllowedIPs= 0.0.0.0/0

Cryptokey Routing



Cryptokey Routing

- Makes system administration very simple.
- If it comes from interface wg0 and is from your friends Bob' tunnel IP address of 192.168.5.17, then the packet definitely came from Bob.
- The iptables rules are plain and clear

Timers: A Stateless Interface for a Stateful Protocol

- As mentioned prior, WireGuard appears “stateless” to user space; you set up your peers, and then it just works.
- A series of timers manages session state internally, invisible to the user.
- Every transition of the state machine has been accounted for, so there are no undefined states or transitions.
- Event based.

Timers

User space sends packet.	• If no session has been established for 120 seconds,send handshake initiation.
No handshake response after 5 seconds.	• Resend handshake initiation.
Successful authentication of incoming packet.	• Send an encrypted empty packet after 10 seconds, if we don't have anything else to send during that time.
No successfully authenticated incoming packets after 15 seconds.	• Send handshake initiation.

Security Design Principle 2: Simplicity of Interface

- The interface appears stateless to the system administrator.
- Add an interface – wg0, wg1, wg2, … – configure its peers, and immediately packets can be sent.
- If it’s not set up correctly, most of the time it will just refuse to work, rather than running insecurely: **fails safe, rather than fails open.**
- Endpoints roam, like in mosh.
- Identities are just the static public keys, just like SSH. Everything else, like session state, connections, and so forth, is invisible to admin.

Demo

Simple Composable Tools

- Since wg(8) is a very simple tool, that works with ip(8), other more complicated tools can be built on top.
- Integration into various network managers:
- OpenWRT
- OpenRC netifrc
- NixOS
- systemd-networkd
- LinuxKit
- Ubiquiti's EdgeOS
- NetworkManager

Simple Composable Tools: wg-quick

- Simple shell script

wg-quick up vpn0

wg-quick down vpn0

- /etc/wireguard/vpn0.conf:

[Interface] Address = 10.200.100.2 DNS = 10.200.100.1

PostDown = resolvconf -d %i

PrivateKey = uDmW0qECQZWPv4K83yg26b3L4r93HvLRcal997IGlEE=

[Peer]

PublicKey = +LRS63OXvyCoVDs1zmWRO/6gVkfQ/pTKEZvZ+CehO1E= AllowedIPs = 0.0.0.0/0

Endpoint = demo.wireguard.io:51820

Security Design Principle 3: Static Fixed Length Headers

- All packet headers have fixed width fields, so no parsing is necessary.
  - Eliminates an entire class of vulnerabilities.
  - No parsers → no parser vulnerabilities.
- Quite a different approach to formats like ASN.1/X.509 or even variable length IP and TCP packet headers.

Security Design Principle 4: Static Allocations and Guarded State

- All state required for WireGuard to work is allocated during config.
- No memory is dynamically allocated in response to received packets.
  - Eliminates another entire classes of vulnerabilities.
  - Places an unusual constraint on the crypto, since we are operating over a finite amount of preallocated memory.
- No state is modified in response to unauthenticated packets.
  - Eliminates yet another entire class of vulnerabilities.
  - Also places unusual constraints on the crypto.

Security Design Principle 5: Stealth

- Some aspects of WireGuard grew out of akernel rootkit project.
- Should not respond to any unauthenticated packets.
- Hinder scanners and service discovery.
- Service only responds to packets with correct crypto.
- Not chatty at all.
  - When there's no data to be exchanged, both peers become silent.

Security Design Principle 6: Solid Crypto

- We make use of Noise Protocol Framework – noiseprotocol.org
  - WireGuard was involved early on with the design of Noise, ensuring it could do what we needed.
  - Custom written very specific implementation of Noise_IKpsk2 for the kernel.
  - Related in spirit to the Signal Protocol.
- The usual list of modern desirable properties you'd want from an authenticated key exchange
- Modern primitives: Curve25519, Blake2s, ChaCha20, Poly1305
- Lack of cipher agility! (Opinionated.)

Security Design Principle 6: Solid Crypto

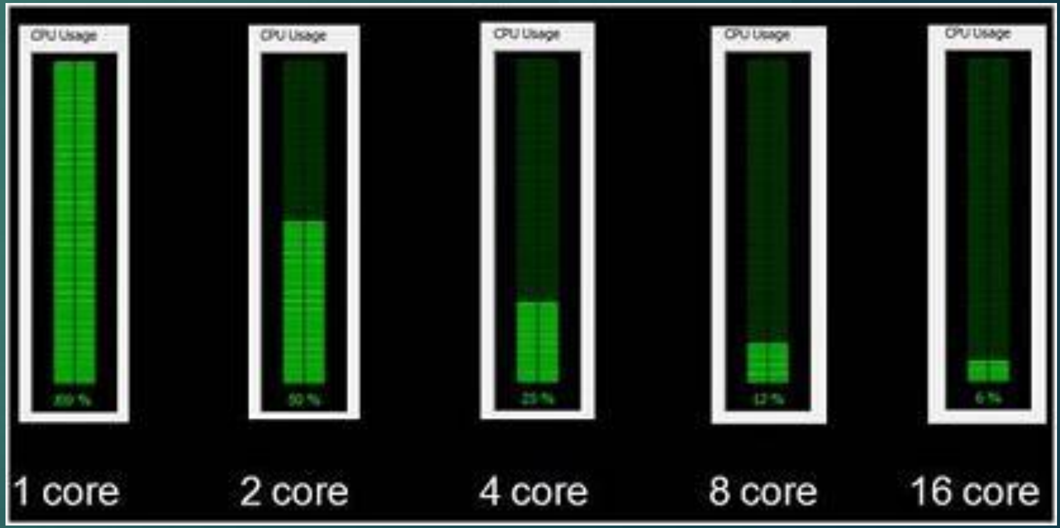
- Strong key agreement & authenticity
- Key-compromise impersonation resistance
- Unknown key-share attack resistance
- Key secrecy
- Forward secrecy
- Session uniqueness
- Identity hiding
- Replay-attack prevention, while allowing for network packet reordering

Crypto Designed for Kernel

- Design goals of guarded memory safety, few allocations, etc have direct effect on cryptography used.
  - Ideally be 1-RTT.
- Fast crypto primitives.
- Clear division between slowpath for ECDH and fastpath for symmetric crypto.
- Handshake in kernel space, instead of punted to userspace daemon like IKE/IPsec.
  - Allows for more efficient and less complex protocols.
  - Exploit interactions between handshake state and packet encryption state.

Multicore Cryptography

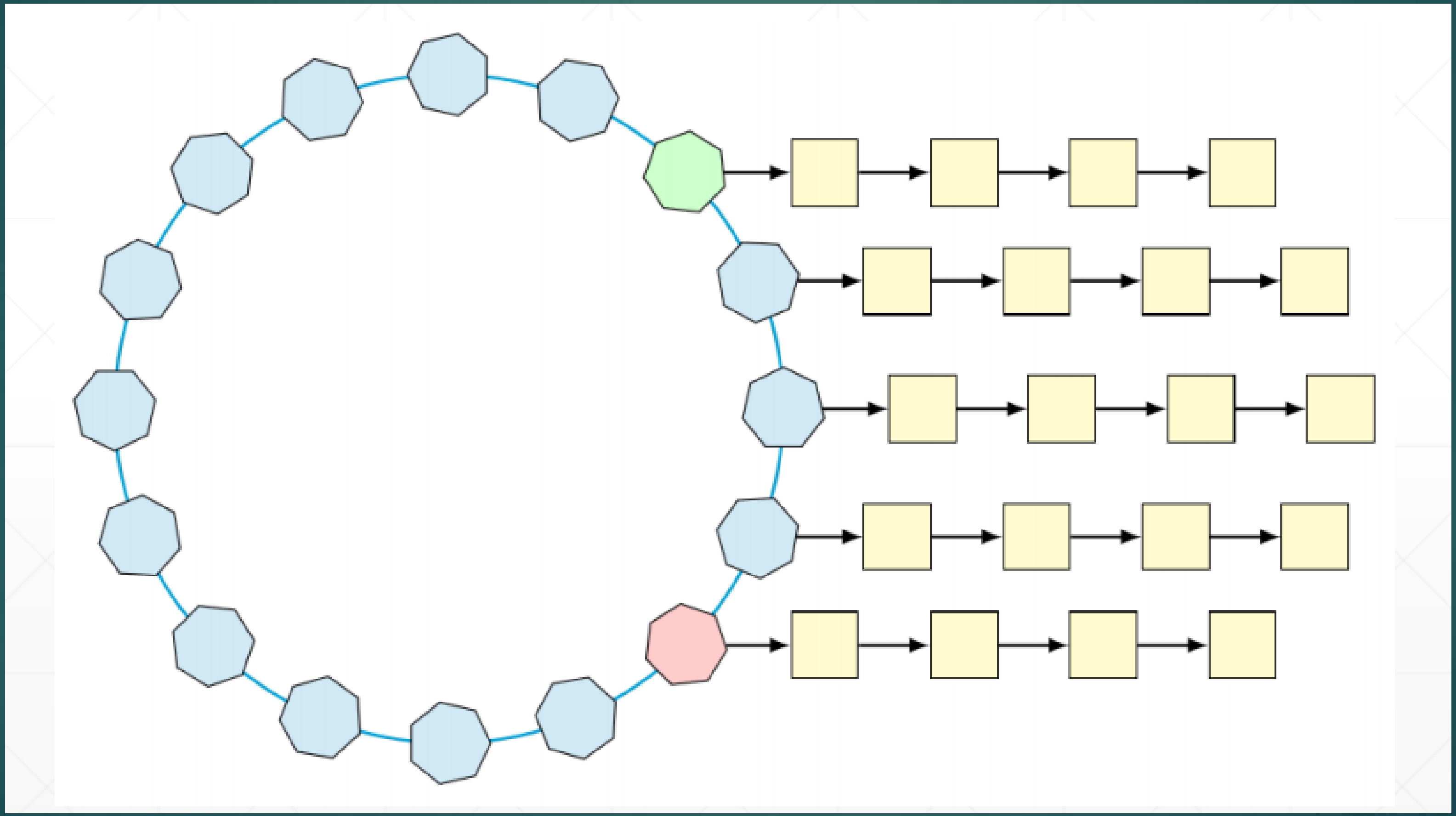
- Encryption and decryption of packets can be spread out to all cores in parallel.
- Nonce/sequence number checking, netif_rx, and transmission must be done in serial order.
- Requirement: fast for single flow traffic in addition to multiflow traffic.
  - Different from usual assumptions.



Multicore Cryptography

- Single queue, shared by all CPUs, rather than queue per CPU
  - No reliance on process scheduler, which tends to add latency when waiting for packets to complete
  - Serial transmission queue waits on ordered completion of parallel queue items
  - Using netif_receive_skb instead of netif_rx to push back on encryption queue
- Bunching bundles of packets together to be encrypted on one CPU results in high performance gains
  - How to choose the size of the bundle?

Multicore Cryptography

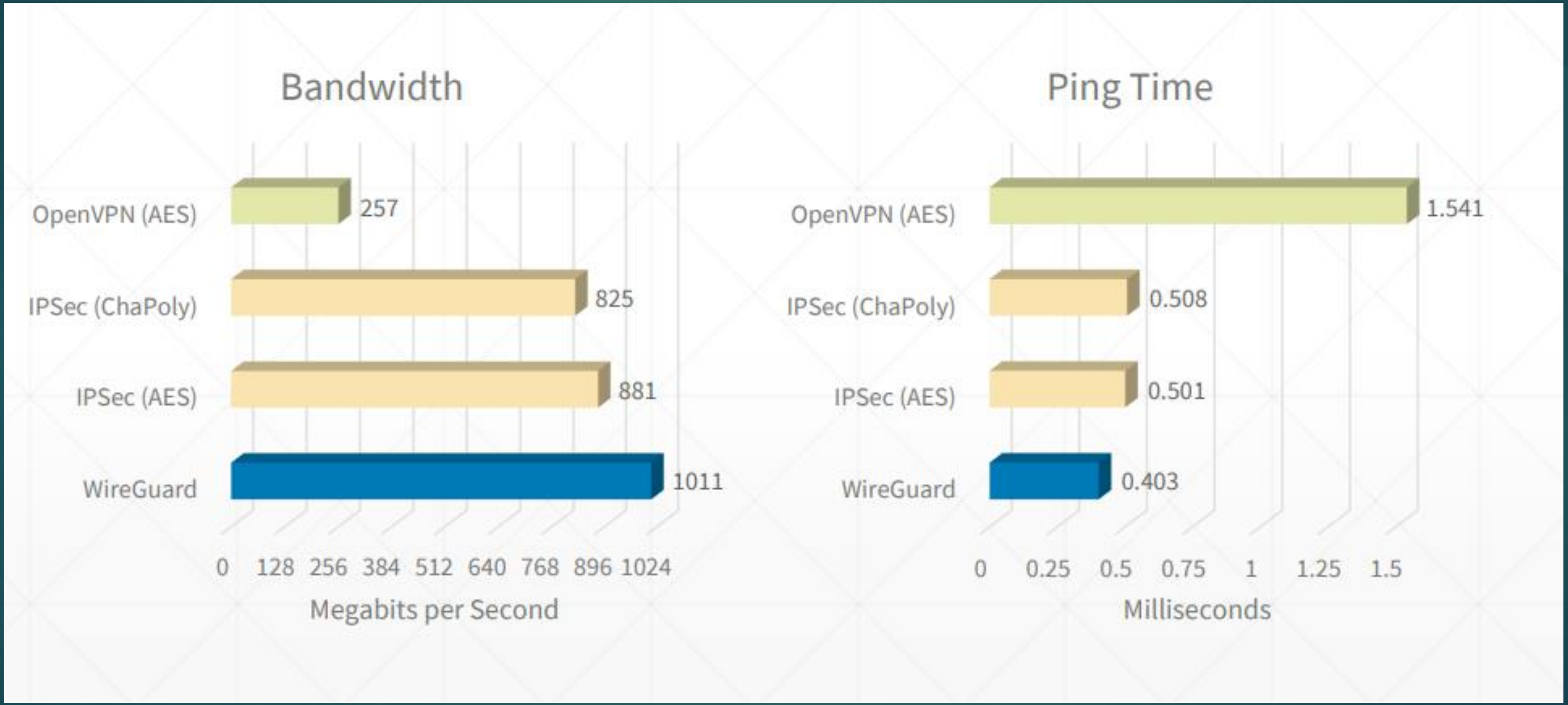


Performance

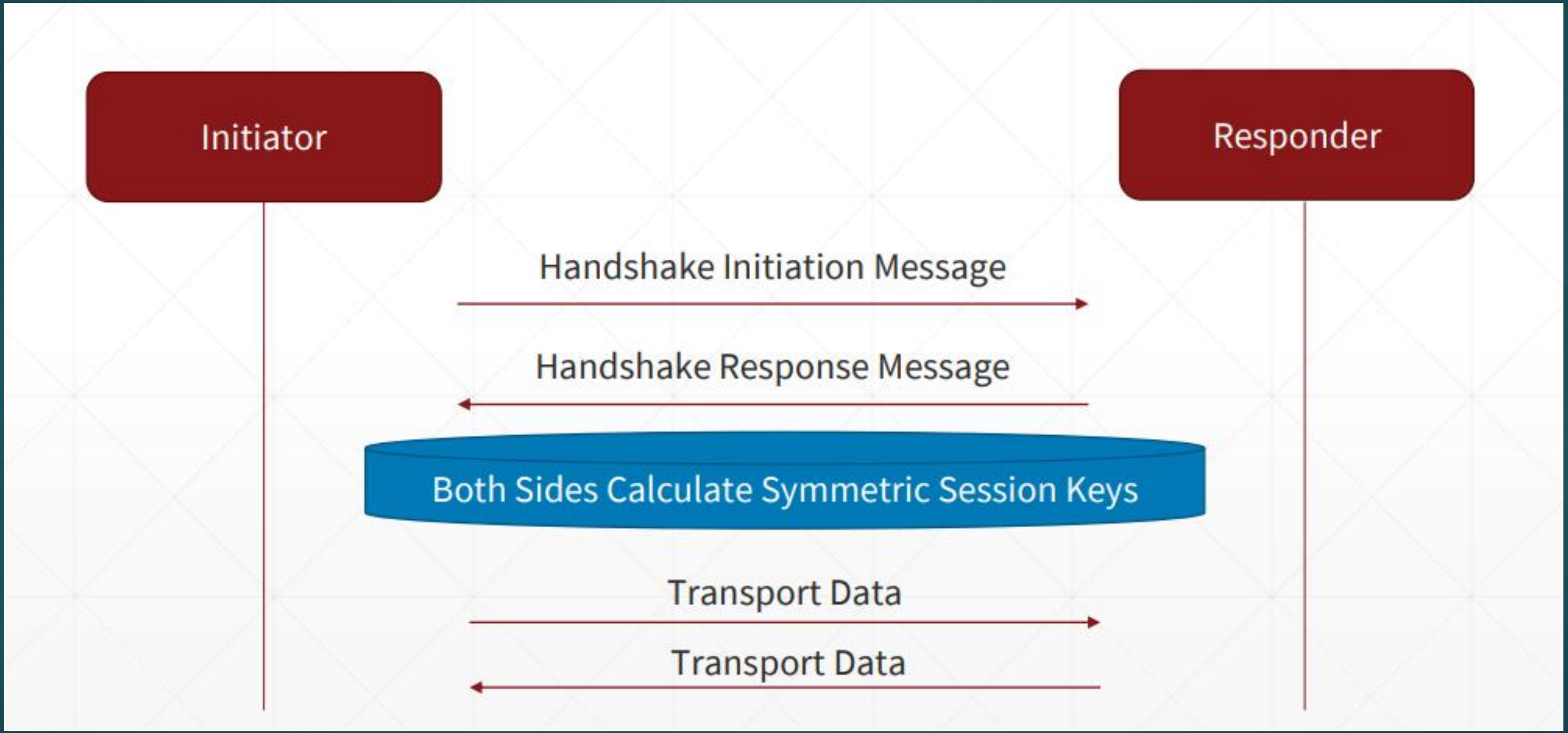
Performance

- Being in kernel space means that it is fast and low latency.
  - No need to copy packets twice between user space and kernel space.
- ChaCha20Poly1305 is extremely fast on nearly all hardware, and safe.
  - AES-NI is fast too, obviously, but as Intel and ARM vector instructions become wider and wider, ChaCha is handedly able to compete with AES-NI, and even perform better in some cases.
  - AES is exceedingly difficult to implement performantly and safely (no cache-timing attacks) without specialized hardware.
  - ChaCha20 can be implemented efficiently on nearly all general purpose processors.
- Simple design of WireGuard means less overhead, and thus better performance.
  - Less code → Faster program? Not always, but in this case, certainly.

Measurements



Confluence of Principles → The Key Exchange



The Key Exchange

- The key exchange designed to keep our principles static allocations, guarded state, fixed length headers, and stealthiness.
- In order for two peers to exchange data, they must first derive ephemeral symmetric crypto session keys from their static public keys.
- Either side can reinitiate the handshake to derive new session keys.
  - So initiator and responder can “swap” roles.
- Invalid handshake messages are ignored, maintaining stealth

The Key Exchange: (Elliptic Curve) Diffie-Hellman Review

private A = random()

public A = derive_public(private A)

private B = random()

public B = derive_public(private B)

ECDH(private A, public B) == ECDH(private B, public A)