WireGuard

FAST, MODERN, SECURE VPN TUNNEL

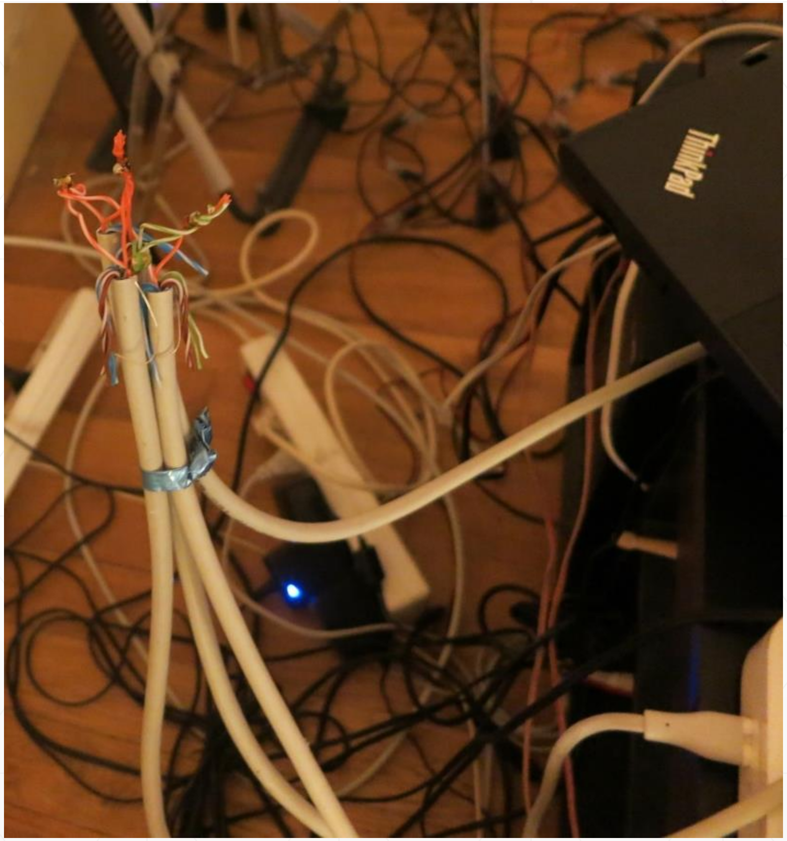
Presented by Jason A. Donenfeld

Who Am I?

- Jason Donenfeld, also known as **zx2c4**.
- Background in exploitation, kernel vulnerabilities, crypto vulnerabilities, and been doing kernel-related development for a long time.
- Motivated to make a VPN that avoids the problems in both crypto and implementation that I've found in numerous other projects.

What is WireGuard?

- Layer 3 secure network tunnel for IPv4 and IPv6.
  - Opinionated. Only layer 3!
- *Designed* for the Linux kernel
  - Slower cross platform implementations also.
- UDP-based. Punches through firewalls.
- Modern conservative cryptographic principles.
- Emphasis on simplicity and auditability.
- Authentication model similar to SSH's `authenticated_keys`.
- Replacement for OpenVPN and IPsec.
- Grew out of a stealth rootkit project.
  - Techniques desired for stealth are equally as useful for tunnel defensive measures.



Blasphemy!

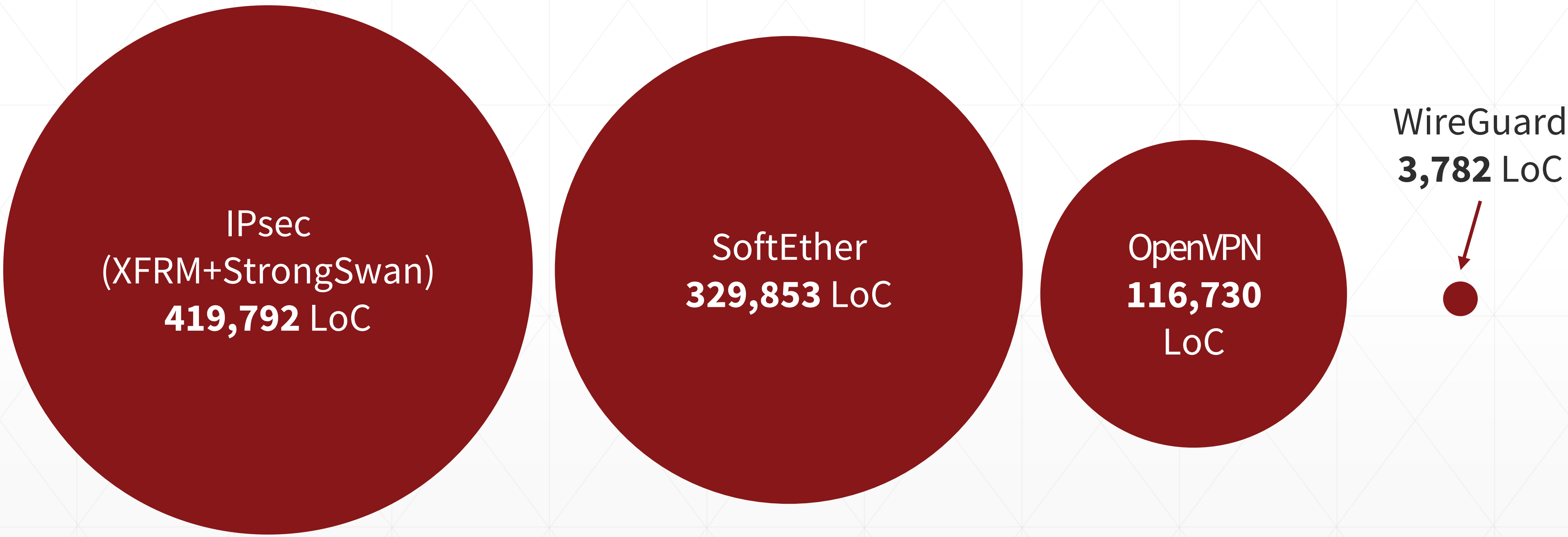
- WireGuard is blasphemous!
- We break several layering assumptions of 90s networking technologies like IPsec.
  - IPsec involves a “transform table” for outgoing packets, which is managed by a user space daemon, which does key exchange and updates the transform table.
- With WireGuard, we start from a very basic building block – the network interface – and build up from there.
- Lacks the academically pristine layering, but through clever organization we arrive at something more coherent.

Easily Auditable

OpenVPN	Linux XFRM	StrongSwan	SoftEther	WireGuard
116,730 LoC Plus OpenSSL!	13,898 LoC Plus StrongSwan!	405,894 LoC Plus XFRM!	329,853 LoC	**3,782 LoC**

Less is more.

Easily Auditable



Simplicity of Interface

- WireGuard presents a normal network interface:

```
# ip link add wg0 type wireguard
# ip address add 192.168.3.2/24 dev wg0
# ip route add default via wg0
# ifconfig wg0 …
# iptables –A INPUT –i wg0 …
```

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

- Everything that ordinarily builds on top of network interfaces – like `eth0` or `wlan0` – can build on top of `wg0`.

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.
- 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.

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 has:
  - A public key
  - A list of associated tunnel IPs
  - Optionally has an endpoint IP and port

Cryptokey Routing

PUBLIC KEY :: IP ADDRESS

Cryptokey Routing

Server Config

```
[Interface]
PrivateKey =
yAnz5TF+lXXJte14tji3zlMNq+hd2rYU
IgJBgB3fBmk=
ListenPort = 41414

[Peer]
PublicKey =
xTIBA5rboUvnH4htodjb6e697QjLERt1
NAB4mZqp8Dg=
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 Config

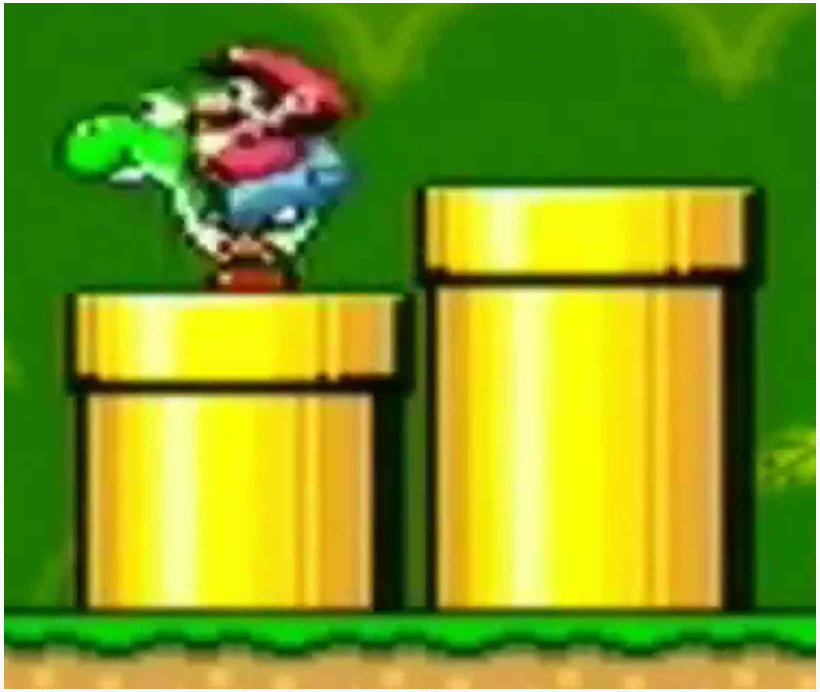
```
[Interface]
PrivateKey =
gI6EdUSYvn8ugXOt8QQD6Yc+JyiZxIhp
3GInSWRfWGE=
ListenPort = 21841

[Peer]
PublicKey =
HIgo9xNzJMWLKASShiTqIybxZ0U3wGLi
UeJ1PKf8ykw=
Endpoint = 192.95.5.69:41414
AllowedIPs = 0.0.0.0/0
```

WireGuard

Cryptokey Routing

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



Demo

Simple API

- Since `wg(8)` is a very simple tool, that works with `ip(8)`, other more complicated tools can be built on top.
- Merge into iproute2 or keep standalone?
- Netlink-based API.
  - Just two commands: `WG_CMD_GET_DEVICE`, `WG_CMD_SET_DEVICE`
  - Set takes device parameters and nested peers with nested allowed IPs
    - Allows userspace to easily fragment massive sets over several separate messages
    - Model is deny-by-default so no races
  - Get returns device parameters and nested peers with nested allowed IPs
    - NLM_F_DUMP
- Roadmap: multicast event notifications for dynamic things.

Easily Composed and Integrated

- Debian’s ifupdown
- OpenWRT/LEDE – core repository
- OpenRC netifrc
- NixOS
- Buildroot
- LinuxKit (from the Docker people)
- EdgeOS / Vyatta / Ubiquiti devices
- Android – runs on the phone in my pocket
- systemd-networkd (WIP)
- NetworkManager (WIP)
- A million trivial shell scripts using wg(8)
- **Packages for 20 different distributions**

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
PrivateKey = uDmW0qECQZWPv4K83yg26b3L4r93HvLRcal997IGlEE=

[Peer]
PublicKey = +LRS63OXvyCoVDs1zmWRO/6gVkfQ/pTKEZvZ+CehO1E=
AllowedIPs = 0.0.0.0/0
Endpoint = demo.wireguard.io:51820
```
```

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.

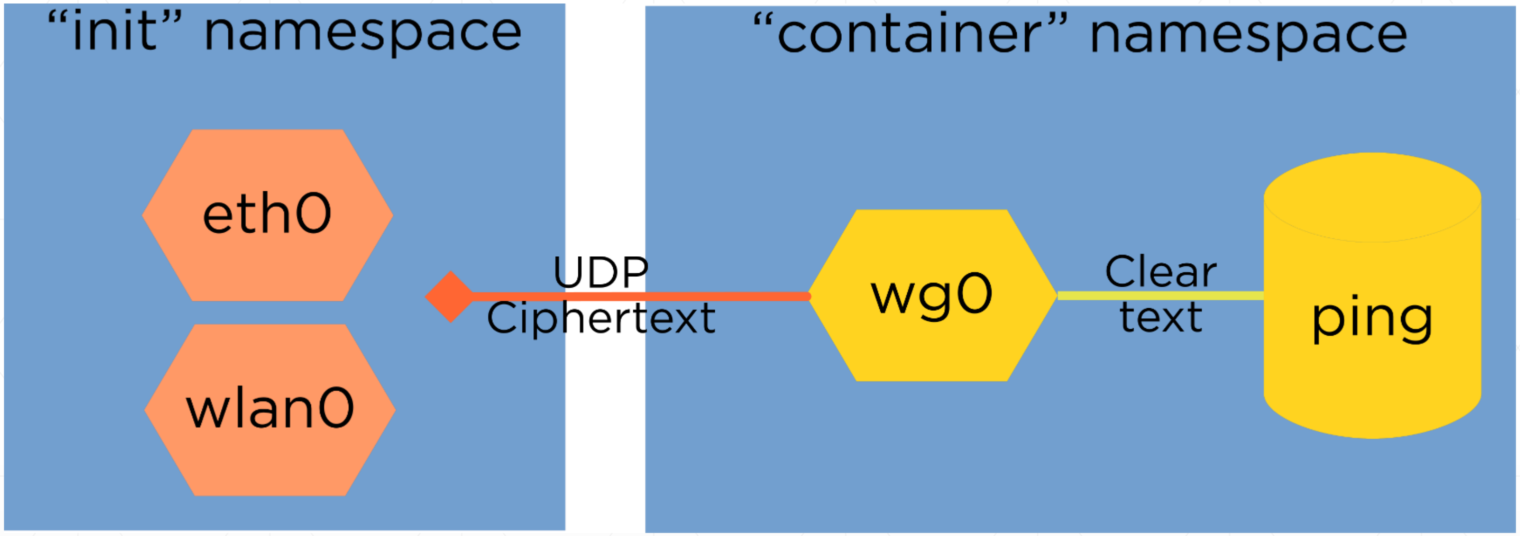
Timers

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.

Network Namespace Tricks

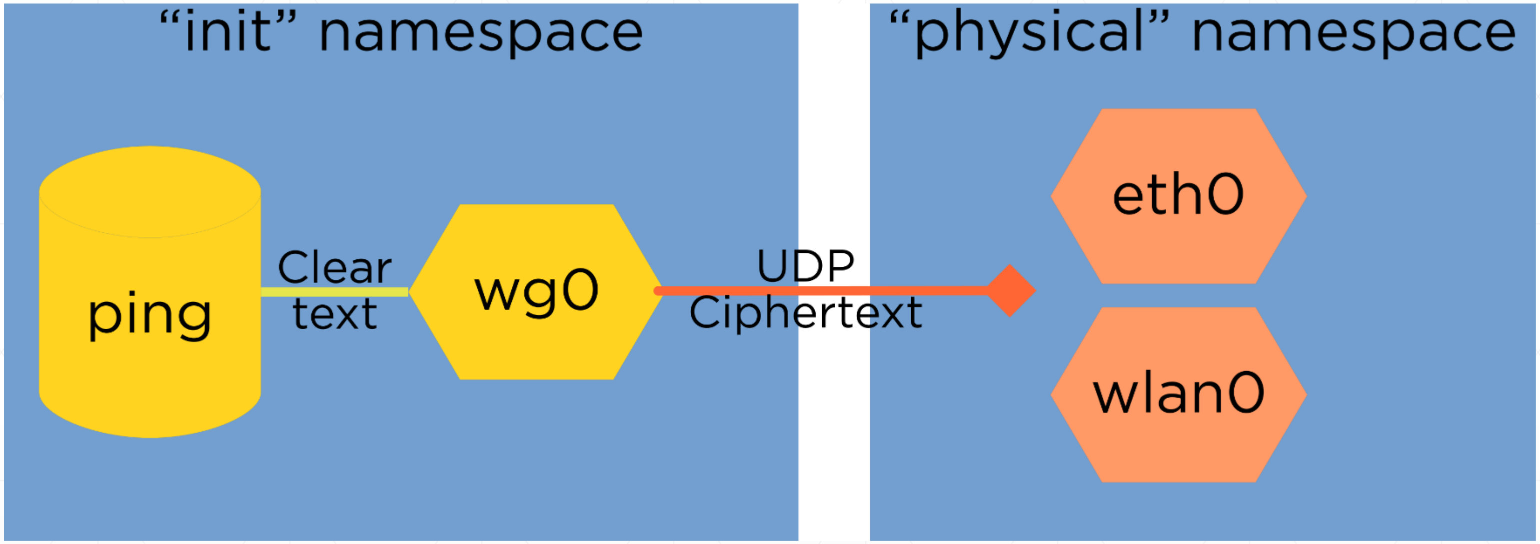
- The WireGuard interface can live in one namespace, and the physical interface can live in another.
- Only let a Docker container connect via WireGuard.
- Only let your DHCP client touch physical interfaces, and only let your web browser see WireGuard interfaces.
- Nice alternative to routing table hacks.
- Means we keep a reference to the source namespace when the `struct net_device` is created.

Namespaces: Containers




```
# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP>
    inet 127.0.0.1/8 scope host lo
17: wg0: <NOARP,UP,LOWER_UP>
    inet 192.168.4.33/32 scope global wg0
```

Namespaces: Personal VPN




```
# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP>
    inet 127.0.0.1/8 scope host lo
17: wg0: <NOARP,UP,LOWER_UP>
    inet 192.168.4.33/32 scope global wg0
```

Policy Routing

- Can set the `fwmark` on outgoing UDP packets (`SO_MARK`)
- Works decently, but `not_oif` / `SO_NOTOIF` would be much simpler:

```
struct flowi fl = {
    .not_oif = dev->ifindex
};
```

or

```
setsockopt(sock, SO_NOTOIF, ifr.ifr_ifindex);
```

- Reduces need for complex ip-rules and suppress_prefix.
- Avoids routing loops.

Stealth

- 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.
- Nice for efficiency on mobile too.

Static Allocations, Guarded State, and Fixed Length Headers

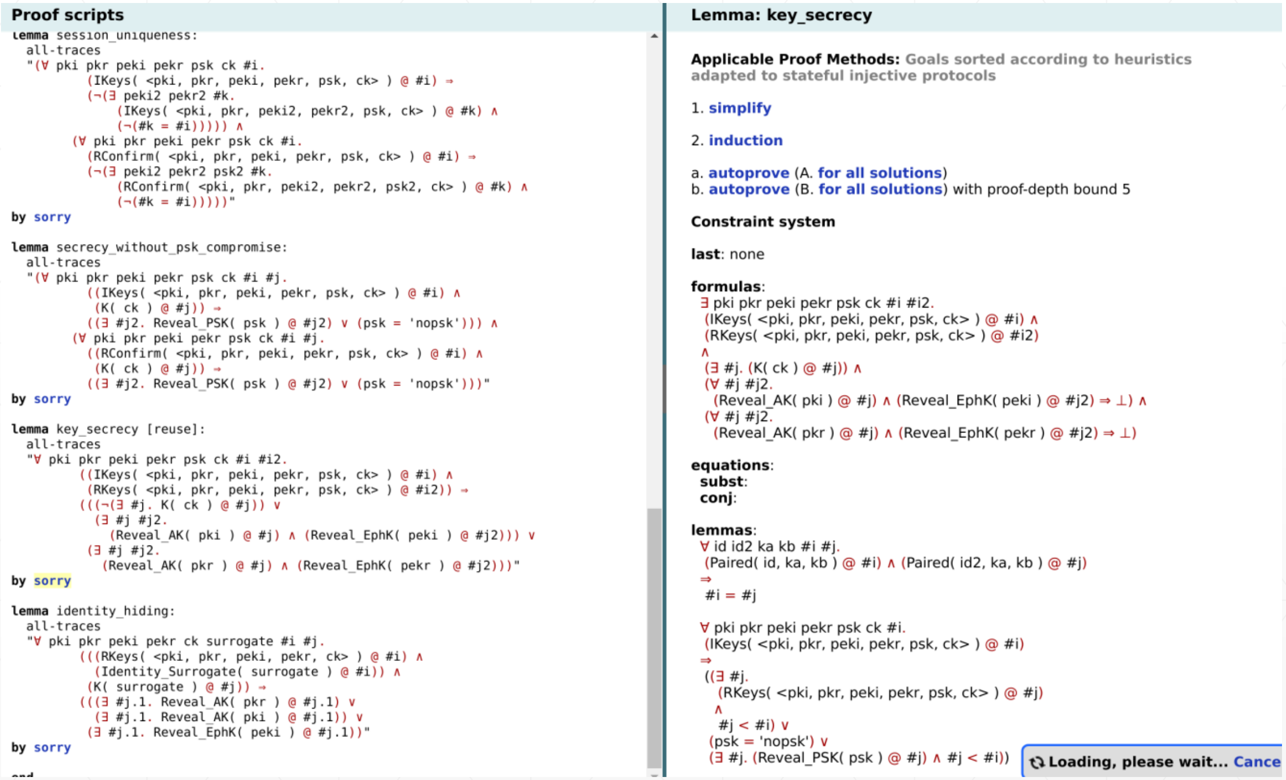
- All state required for WireGuard to work is allocated during config.
- No memory is dynamically allocated in response to received packets.
  - Eliminates entire classes of vulnerabilities.
- All packet headers have fixed width fields, so no parsing is necessary.
  - Eliminates *another* entire class of vulnerabilities.
- No state is modified in response to unauthenticated packets.
  - Eliminates *yet another* entire class of vulnerabilities.

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 (workqueues) for ECDH and fastpath for symmetric crypto.
- Handshake in kernelspace, 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.

Formal Symbolic Verification

- The cryptographic protocol has been formally verified using Tamarin.



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.

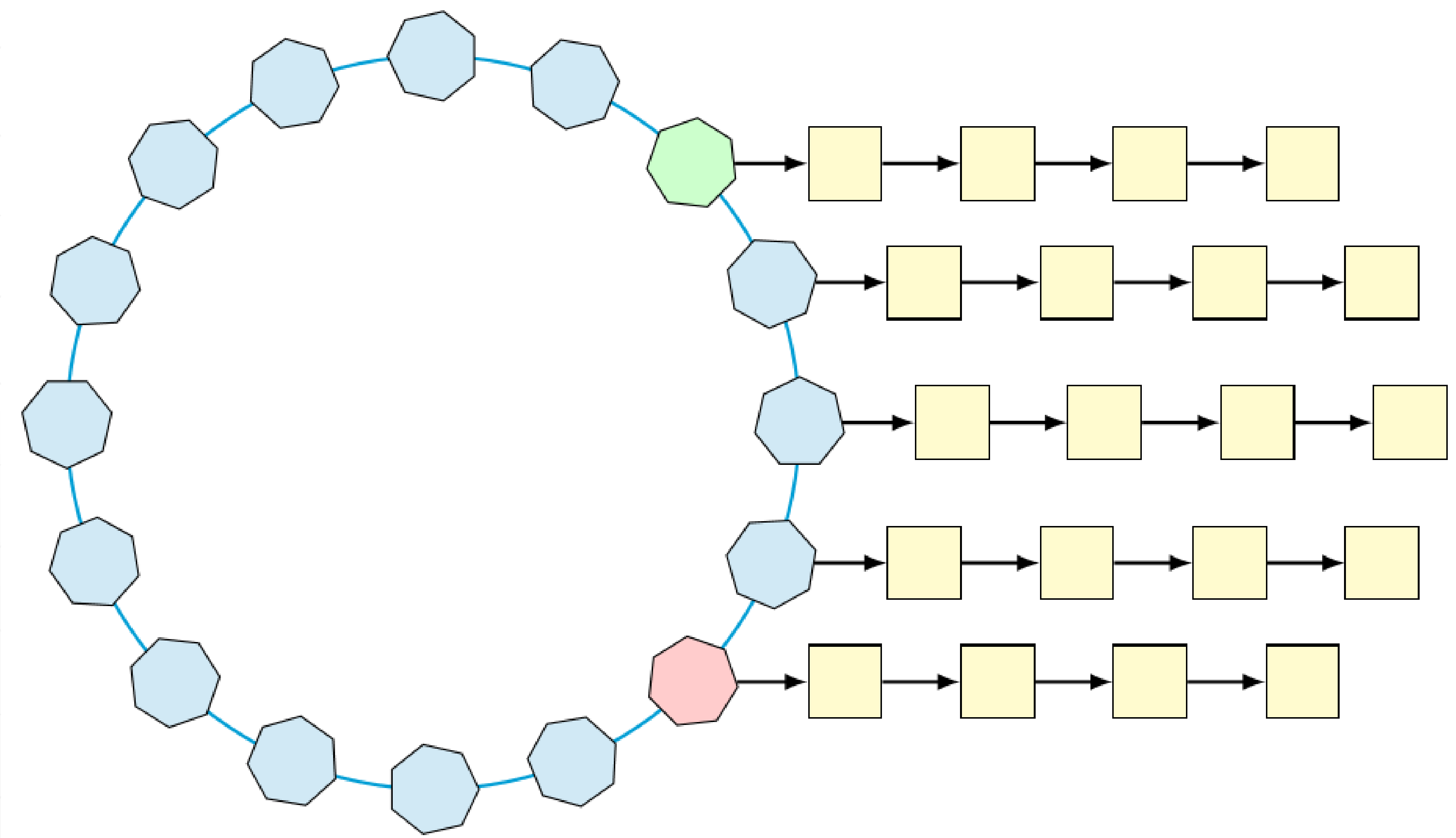
Multicore Cryptography

- Parallel encryption queue is multi-producer, multi-consumer
  - Lockless algorithms?
  - Lockless linked list is difficult, but lockless ring buffer is more common
- 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?

Generic Segmentation Offload

- By advertising that the `net_device` suppports GSO, WireGuard receives massive "super-packets" all at the same time.
- WireGuard can then split the super-packets by itself, and bundle these to be encrypted on a single CPU all at once.
- Each bundle is a linked list of skbs, which is added to the ring buffer queue.

Multicore Cryptography



Multicore Cryptography

- How to determine queue lengths?
- One approach is to just have a fixed queue length, that isn't overly big.
- Queues could alternatively use `struct dql`, or full on `fq_codel`.
- If `fq_codel`, use via qdisc, or directly like certain wifi drivers?
- Fairness between peers is consideration.
- Advantage of `IFF_NO_QUEUE` is that we can return errno to userspace directly.
  - `-ENOKEY, -EDESTADDRREQ, -EPROTONOSUPPORT`
  - There's ICMP for this too, though.
    - NAT is still an issue.

In-band Messaging

- Some folks wish to send in-band configuration messages.
  - Dynamic IP addresses, other horrible things.
  - New fangled post-quantum key exchanges.
  - Other monstrous things too!
- What situations necessitate in-band control messages?
  - How much can be done out-of-band or statically, during the actual key exchange step?

In-band Messaging

- Three approaches toward in-band messaging:
  1. AF_WIREGUARD
     - Elegant, sleek, obvious
     - Hard to justify adding a new AF
     - Host of interesting unforeseeable possibilities and uses
  2. Netlink Events
     - More typical way of doing it these days
     - Unintrusive
     - Reinforces it being for control messages, not for real data
  3. Not supporting it
     - Keep doing things out of band!
     - Simpler, cleaner

Sticky Sockets

- WireGuard listens on all addresses, but manages to always reply using the right source address.
- Caching of destination address and interface of incoming packets, but ensures that this stickiness isn’t too sticky.
- Does the right thing every time – interface disconnects, routes change, etc.
- Actually maps mostly nicely to possible semantics of `IP_PKTINFO`, so userspace implementations can do this too, sort of.

Secret Handling

- Extensive use of `memzero_explicit`.
- Much crypto-related code in the kernel forgets or does not care.
  - KTLS!
- Netlink is very problematic, since it uses skbs.
  - New skb flag? `SKB_ZERO_ON_FREE`?

Crypto API Improvements

- WireGuard uses its own internal crypto API and primitives.
- Road ahead for working these enhancements into kernel's crypto API.
- Direct function calls, without abstraction layer.
- Advanced protocols need to change key frequently.
- Avoid allocations.
- WIP: formally verified implementations from INRIA.

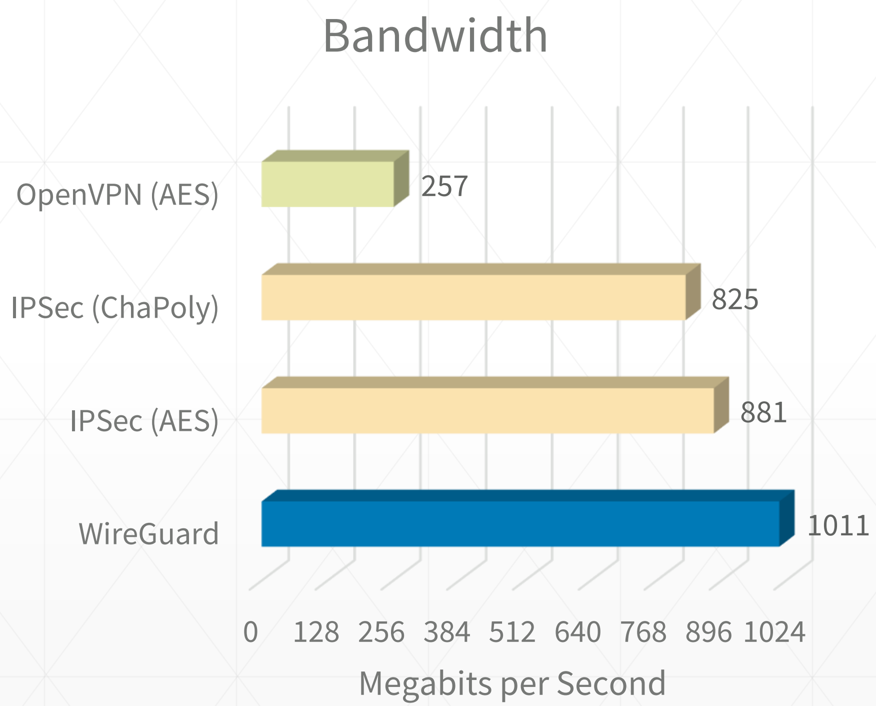
Crypto API: Batching of FPU Context

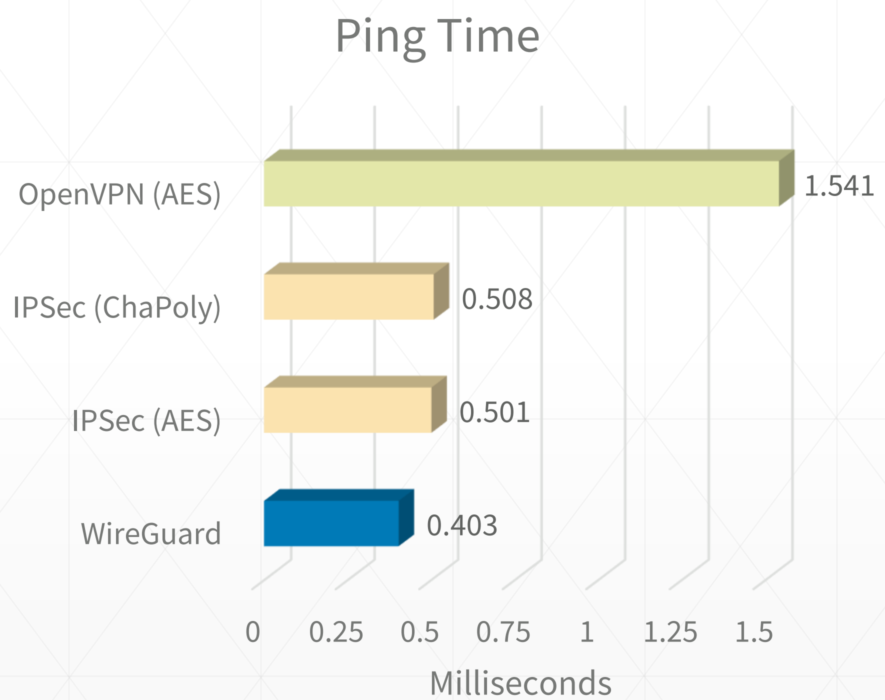
- Saving and restoring FPU registers multiple times is inefficient.
- Save these once per thread, by hoisting calls out `kernel_fpu_begin` outside encryption loops.
- Straightforward approach, but lazy restoration might be cleaner and require less state passing.

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.

Performance





WireGuard

Continuous Integration

- Extensive test suite, trying all sorts of topologies and many strange behaviors and edge cases.
- Every commit is tested on every kernel.org kernel (and a few more), and built and run fresh in QEMU
- Tests on x86_64, ARM, AArch64, MIPS

build.wireguard.com

Linux 4.14-rc8 (aarch64)	Success
Linux 4.14-rc8 (arm)	Success

```
Show build details.

   WireGuard Test Suite on Linux 4.14.0-rc8 armv7l

[+] Mounting filesystems...
[+] Module self-tests:
 *  routing table self-tests: pass
 *  nonce counter self-tests: pass
 *  curve25519 self-tests: pass
 *  chacha20poly1305 self-tests: pass
 *  blake2s self-tests: pass
 *  ratelimiter self-tests: pass
[+] Enabling logging...
[+] Launching tests...
[+] ip netns add wg-test-44-0
[+] ip netns add wg-test-44-1
[+] ip netns add wg-test-44-2
```

Linux 4.14-rc8 (mips)	Success
Linux 4.13.11 (x86_64)	Success
Linux 4.9.60 (x86_64)	Success
Linux 4.4.96 (x86_64)	Success
Linux 4.1.45 (x86_64)	Success

WireGuard

Upstream Roadmap

- Multicast Netlink events.
- Maybe in-band messages.
- **Biggest blocker is crypto API.**
- Eyeing beginning of next year for initial [PATCH] post.
- Already integrated into many distributions and sees regular testing on network intense projects like LEDE/OpenWRT and LinuxKit.
- Commercial VPN providers already using it.
- Regular snapshot releases are being made.
- Now is time to start soliciting upstream feedback.

- Available now for all major distros: wireguard.com/install
- Build it directly into the kernel or compile it as a module.
- Peer-reviewed paper published in NDSS 2017, available at: wireguard.com/papers/wireguard.pdf
- `$ git clone https://git.zx2c4.com/WireGuard`
- wireguard@lists.zx2c4.com
  lists.zx2c4.com/mailman/listinfo/wireguard
- `#wireguard` on Freenode
- **STICKERS FOR EVERYBODY:**
  lists.zx2c4.com/pipermail/wireguard/2017-May/001338.html
- Plenty of work to be done: looking for interested devs.

Jason Donenfeld

- Personal website: www.zx2c4.com
- Email: Jason@zx2c4.com
- Company: www.edgesecurity.com

EDGESECURITY