>>> VPNs mit Wireguard + nftables
>>> Fast, Modern, Secure VPN Tunnel

Name: Frederik Schwan[†]
Date: May 28, 2019

[†]frederik.schwan@linux.com

>>> Überblick

1. Wireguard
2. Wireguard/OpenVPN Vergleich
3. Live Demo Wireguard
4. nftables
5. Live Demo nftables

>>> Motivation

Peering

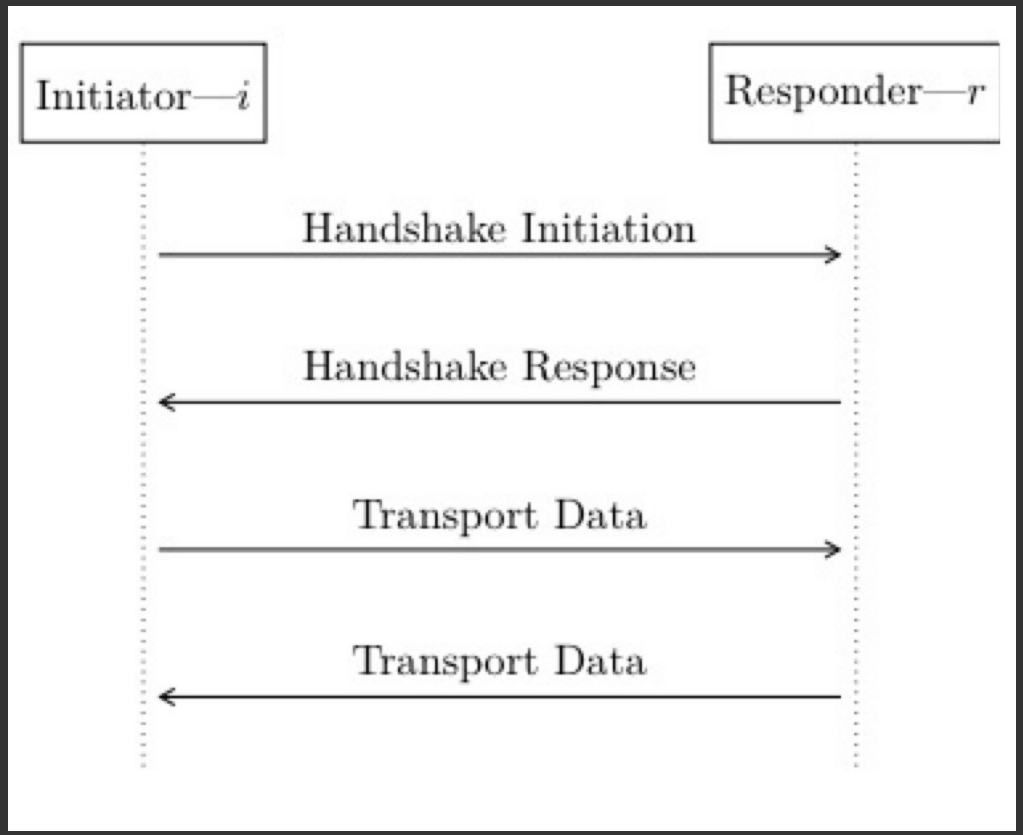
>>> Autor

* Jason Donenfeld
* Finden von Sicherheitslücken, Kernel & Krypto Vulnerabilities, Entwicklung mit Bezug zum Linux Kernel
* Autor von unix pass
* Auch als zx2c4 bekannt

>>> Wireguard Fakten

* Layer 3
* UDP Transport
* pk/sk Krypto
* Einfache Konfiguration
* benutzt elliptische Kurven (ChaCha20, Poly1305 und Curve25519 für ECDH)
* Aktuell 2-3 aktive Entwickler inkl. zx2c4

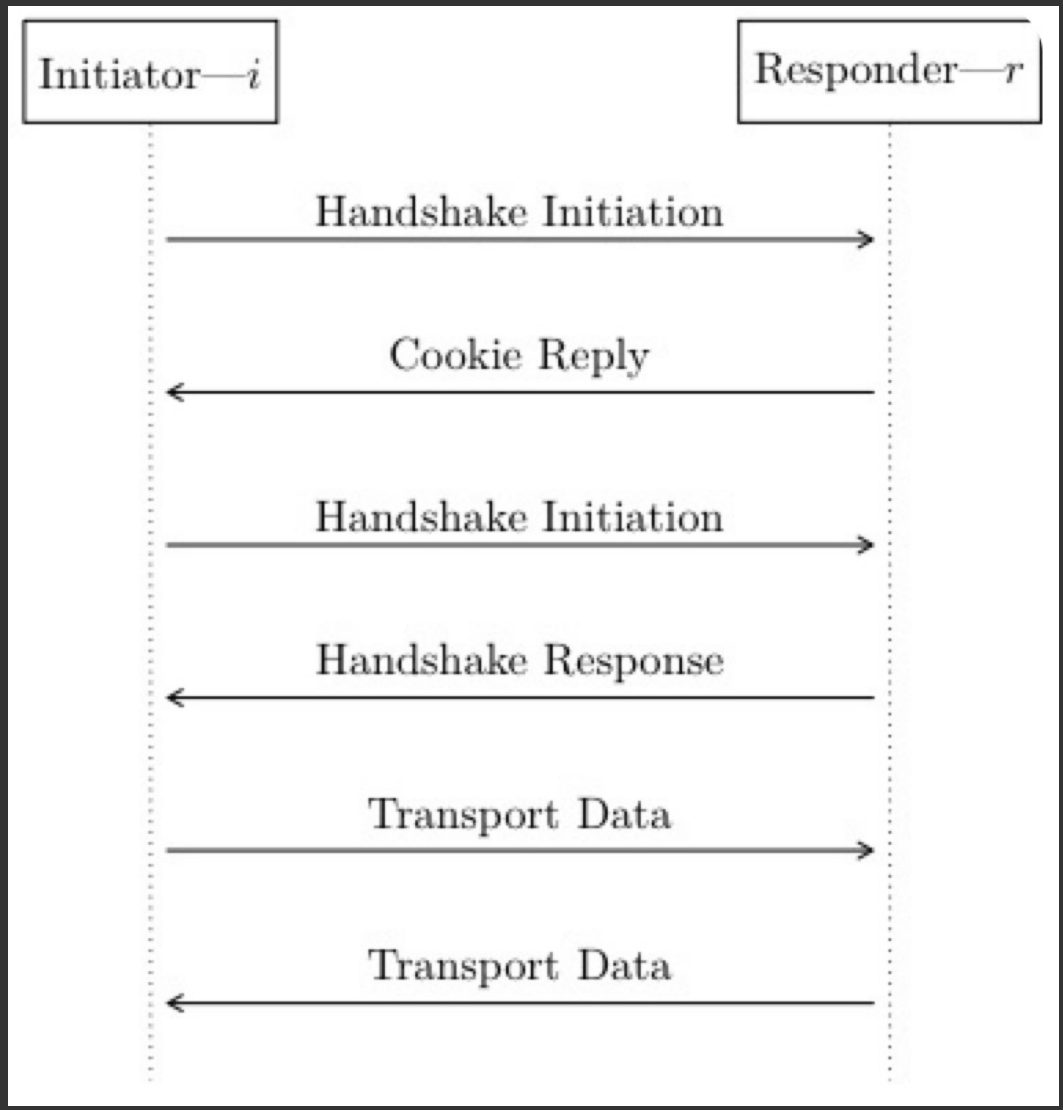
>>> Wireguard Protokoll



[1]

[1] https://www.wireguard.com/papers/wireguard.pdf

>>> Wireguard Protokoll DOS



[2]

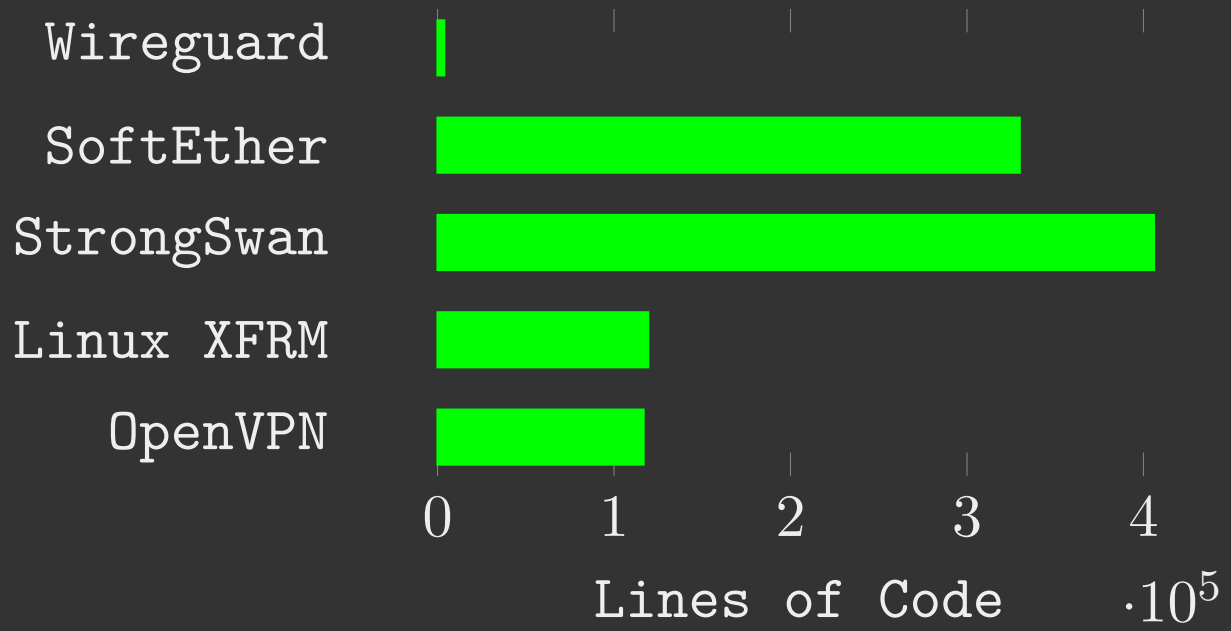
[2] https://www.wireguard.com/papers/wireguard.pdf

>>> OpenVPN Protokoll

* OpenSSL Handshake:
  * 2 RTT
  * Aushandlung von Ciphers
* 1 RTT für Control
* Control Channel händelt auch IP/DNS Konfiguration
* Komplexität von SSL + VPN Logik

>>> Wireguard Komplexität

VPN	LOC	Misc. [3]
WireGuard	3.771	+ Crypto
SoftEther	329.853	
StrongSwan	405.894	+ XFRM
Linux XFRM	119.363	+ StrongSwan
OpenVPN	116.730	+ OpenSSL



[3] https://www.wireguard.com/papers/wireguard.pdf

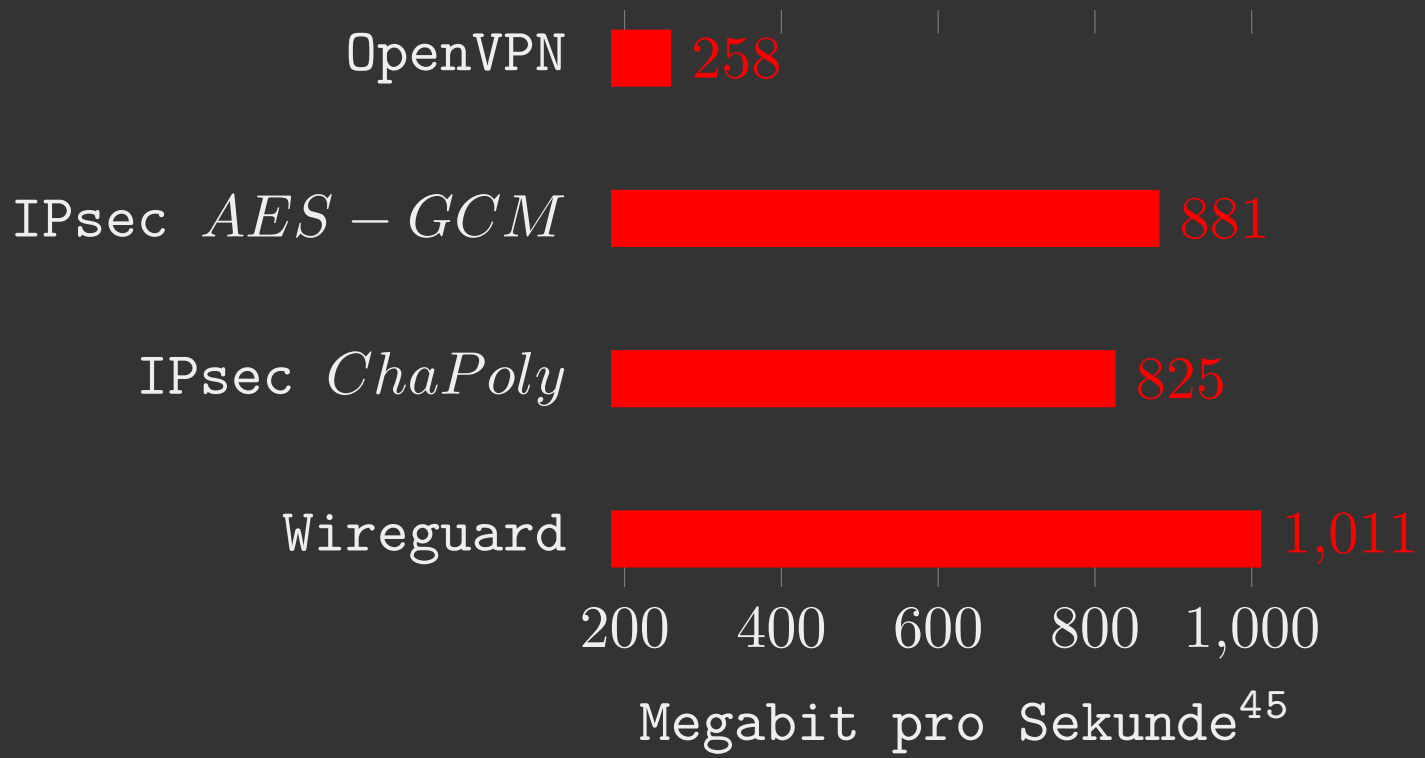
>>> Benchmarks

Code aktuell unoptimiert!

Roadmap:

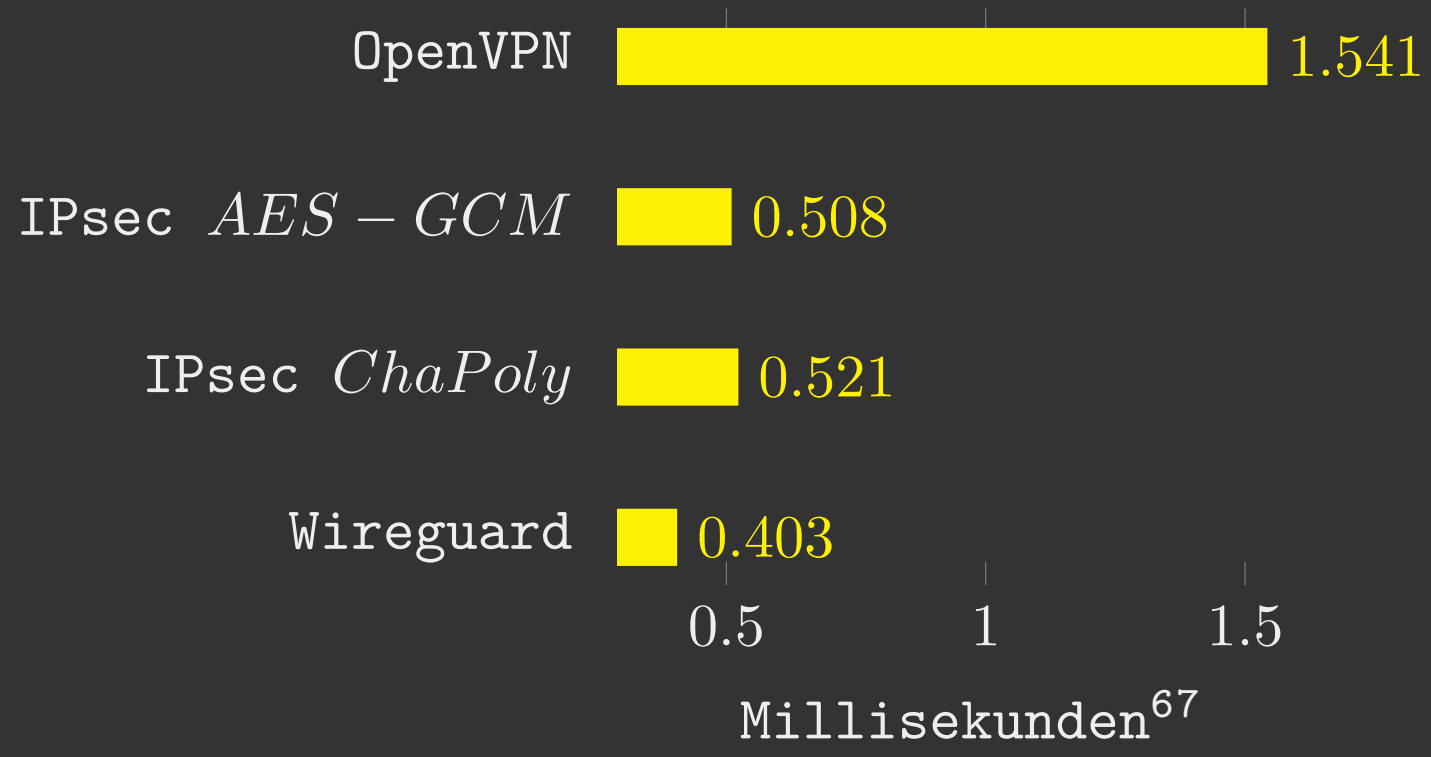
* Unterstützung Generic Receive Offload (GRO)
* Entfernen von Locking in Queues
* Automatische Skalierung über CPU-Kerne
* Lokalität der Pakete in der CPU
* Integration in Network Scheduler (qdisc/fq_codel/dql)

>>> Performance Vergleich Durchsatz



[4] https://www.wireguard.com/papers/wireguard.pdf
[5] i7-3820QM & i7-5200U mit Linux 4.6.1

>>> Performance Vergleich Ping



[6] https://www.wireguard.com/papers/wireguard.pdf
[7] i7-3820QM & i7-5200U mit Linux 4.6.1

>>> Nachteile

* keine dynamischen Adressen im Wireguard Netz
* kein TCP Support
* Kernel Implementierung unflexibel
* Debian -> Sid/Unstable

>>> Live Demo

>>> Fakten

* erstmals 2008 Präsentiert
* enthalten seit Kernel 3.13 ~01/14
* Userspace Tool und Subsystem haben gleichen Namen
* Kompatiblitätslayer iptables Userspace → nftables Kernel Subsystem existiert
* Default für Debian Buster

>>> Features

* iptables, ip6tables, arptables, ebtables in 1
* nft ermöglicht übersichtliches Skripten

>>> nftables Beispiele

iptables:

* iptables -A OUTPUT -d 1.2.3.4 -j DROP

nftables:

* nft add rule ip filter output ip daddr 1.2.3.4 drop

>>> nftables Beispiele

iptables:

* ip6tables -A INPUT -p tcp -m multiport --dports 23,80,443 -j LOG
* ip6tables -A INPUT -p tcp -m multiport --dports 23,80,443 -j ACCEPT

nftables:

* nft add rule inet filter input tcp dport {telnet, http, https} log accept

>>> nftables Beispiele

iptables:

* ip6tables -A INPUT -p icmpv6 --icmpv6-type neighbor-solicitation -j ACCEPT
* ip6tables -A INPUT -p icmpv6 --icmpv6-type echo-request -j ACCEPT

nftables:

* nft add rule ip6 filter input icmpv6 type {nd-neighbor-solicit, echo-request} accept

>>> SSH Bruteforce

iptables:

* `iptables -I INPUT -p tcp --dport 22 -i eth0 -m state --state NEW -m recent --set`
* `iptables -I INPUT -p tcp --dport 22 -i eth0 -m state --state NEW -m recent --update --seconds 3600 --hitcount 20 -j DROP`

nftables:

* `nft add rule ip filter input tcp dport 22 ct state new meter ssh-meter4 { ip saddr limit rate 20/hour } accept`
* `nft add rule ip6 filter input tcp dport 22 ct state new meter ssh-meter6 { ip6 saddr limit rate 20/hour } accept`
* `... drop`

`nft list meter filter ssh-meter4`

>>> File Beispiel

```
#!/usr/bin/nft -f
flush ruleset

table inet filter {
  chain input {
   type filter hook input priority 0
   tcp dport 22000 accept
  }
}
table ip nat {
  chain postrouting {
    type nat hook postrouting priority 0
    ip saddr 192.168.178.0/24 snat 1.2.3.4
  }
}
```

>>> Live Demo