WireGuard

Einfache, robuste und sichere VPNs

Wer bin ich?

- Alissa Gerhard
  (sie/ihr)

- 2014-2019: Forscherin zu Future Internet (TU KL, KIT)
- Seit 2020: Systemintegratorin bei Port Zero

Port Zero

Agenda

Überblick

Quick Start

Protokoll

Performance

Administration

Überblick
Quick Start
Protokoll
Performance
Administration

Wireguard
08.06.2023

GPN21

WireGuard

- Vergleichsweise neues VPN-Protokoll
- UDP-basiert, Peer-to-Peer, Kryptographie-zentrierter Ansatz
- Verbindungslos
  - Unterstützt Roaming
  - Überlebt Verbindungsabbrüche
  - Transparente Session-Neuverhandlung
- Minimiert Angriffsfläche
  - Unsichtbar in Port-Scans
  - DoS-Schutz
  - Replay-Schutz
- Integriert in Linux-Kernel seit 2020

Cryptokey Routing

- Peers werden anhand ihres statischen Public Keys identifiziert
  - Jede Nachricht ist vom Sender authentifiziert (PubKey or Session Key)
- Jeder Peer pflegt eine Tabelle von Peers:

PubKey / ID	Erlaubte IPs	Internet-Endpunkt
`TrMv..X0=`	`10.0.0.2/32`	
`gN65..EA=`	`10.0.0.3/32`	`203.0.113.20`

- Erlaubte IPs dienen als Routing-Tabelle
- Internet-Endpunkt wird bei jeder empfangenen authentifizierten Nachricht aktualisiert

Überblick
Quick Start
Protokoll
Performance
Administration

Quick Start

Wireguard
08.06.2023

Installation und Konfiguration: Linux

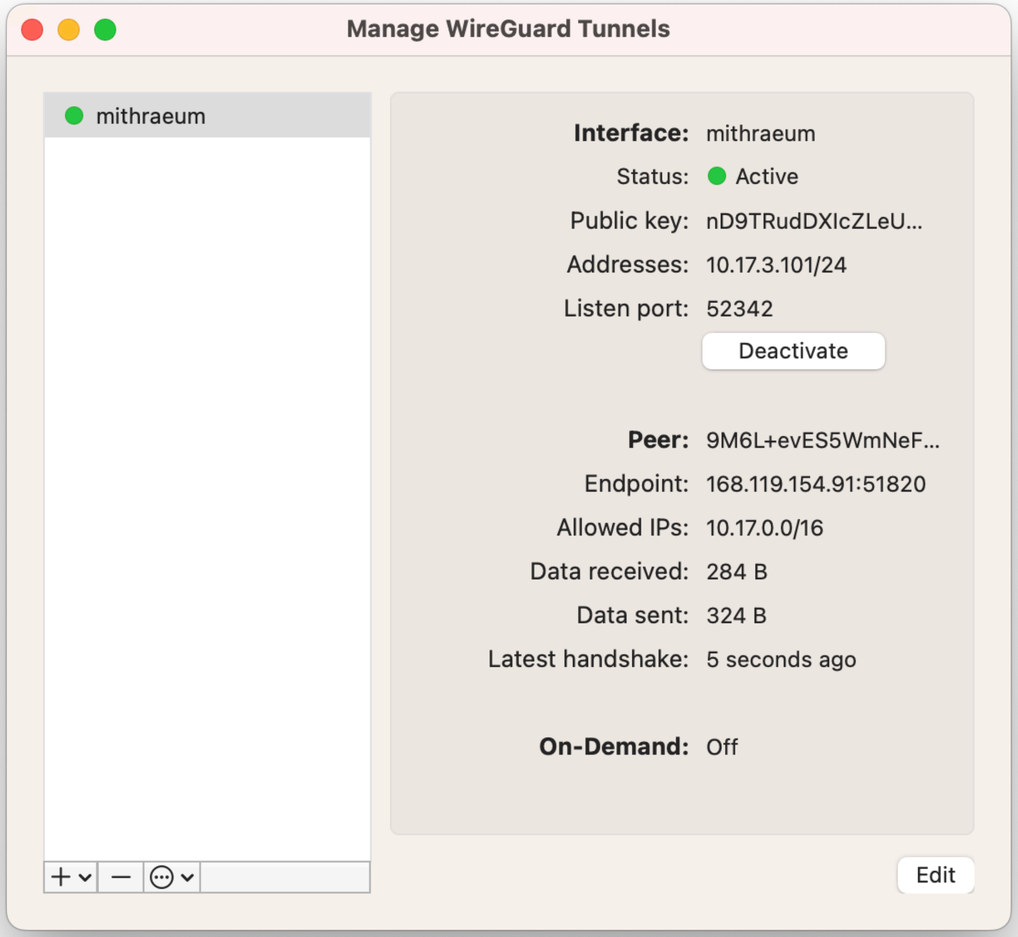
- Benötigt Kernel 5.6 oder höher
- Entsprechendes Paket der Distro (oft **`wireguard-tools`**)
- Konfiguration vollständig mittels “**`ip`**” und “**`wg`**” tools möglich

- Üblicherweise wird der bash-wrapper “**`wg-quick`**” verwendet
  - Konfigurationsdatei in **`/etc/wireguard/wg0.conf`**
    - (Dateiname entspricht späterem Namen des Interfaces)
  - Start/Stopp mit systemd-Service **`wg-quick@wg0.service`**

Installation und Konfiguration: Andere

macOS, iOS, Android, Windows

- Offizieller grafischer Client von **`wireguard.com`**
- Verwalten und Laden der Konfigurationen in GUI
- Weitere Optionen verfügbar



Erzeugung der kryptographischen Keys

- Eingebaute Funktionalität zum Erzeugen der Private/Public Keys

```
$ wg genkey
4EcUDsTnuR0lFaDmBYun3K7gCYPd5NALXJHXPaX7iVA=
$ echo 4EcUDsTnuR0lFaDmBYun3K7gCYPd5NALXJHXPaX7iVA= | wg pubkey
dBVBpGn1HxpJWL1bCh/XoT8E7ogRLjPPptGp91xWeBc=
```

- Nutzt unix-üblich stdin/stdout zum einfachen Einbinden in Workflows
- Nicht parametrisierbar: WG schreibt Kryptographie vollständig vor

Erzeugung der kryptographischen Keys

- Eingebaute Funktionalität zum Erzeugen der Private/Public Keys

```
$ wg genkey > wg-private.key
$ wg pubkey < wg-private.key > wg-public.key
```

- Nutzt unix-üblich stdin/stdout zum einfachen Einbinden in Workflows
- Nicht parametrisierbar: WG schreibt Kryptographie vollständig vor

Einfaches Client-Server Setup

- Clients können mit dem Server kommunizieren
- Server kann Client nur erreichen, nachdem sich Client beim Server gemeldet hat

server

```
[Interface]
Address = 10.0.0.1/24
PrivateKey = yGdqWmv16VvbRbLLi1scAJ2Zz8J6ylKbrnLMNCUEVVc=
ListenPort = 51820

[Peer]
PublicKey = dBVBpGn1HxpJWL1bCh/XoT8E7ogRLjPPptGp91xWeBc=
AllowedIPs = 10.0.0.2/32

[Peer]
PublicKey = l1z+2WEg+J1ADnZDJnj3n8zIXvkYPlqEgGkpUnTkqVQ=
AllowedIPs = 10.0.0.3/32
```

client

```
[Interface]
Address = 10.0.0.2/24
ListenPort = 51820
PrivateKey = 4EcUDsTnuR0lFaDmBYun3K7gCYPd5NALXJHXPaX7iVA=

[Peer]
PublicKey = CnmIgy79BdTSnNJ1sP70BUkl+CBl/LXbQGKI048bHQE=
Endpoint = 108.51.100.36:51820
AllowedIPs = 10.0.0.1/24
```

Weiterleitung im Server

- Erlaubt Kommunikation zwischen Clients
- Erlaubt es Clients, in ein eventuell vorhandenes, internes Netz des Servers zu kommunizieren

server

```
[Interface]
Address = 10.0.0.1
PrivateKey = [redacted]v16VvbRbLLi1scAJ2Zz8J6ylKbrnLMNCUEVVc=
ListenPort = 51820
PostUp = iptables -A FORWARD -i %i -j ACCEPT; iptables -A
-o %i -j ACCEPT
PostDown = iptables -D FORWARD -i %i -j ACCEPT; iptables
-D -o %i -j ACCEPT

# Managing sysctl variable is preferably done outside of
wireguard though
PostUp = sysctl -w net.ipv4.ip_forward=1

[Peer]
...
```

client

```
[Interface]
Address = 10.0.0.2/24
ListenPort = 51820
PrivateKey = 4EcUDsTnuR0lFaDmBYu[redacted]LYPd5NALXJHXPaX7iVA=

[Peer]
PublicKey = CnmIgy79BdTSn[redacted]0BUkl+CBl/LXbQGKI048bHQE=
Endpoint = 108.51.100.36:[redacted]
AllowedIPs = 10.0.0.1/24
```

Voll-Tunnel VPN

- Clients können nicht untereinander kommunizieren
- Kommunikation von Clients ins Internet möglich; Server nutzt NAT
- Client nutzt den Server als Voll-Tunnel

server

```
[Interface]
Address = 10.0.0.1
PrivateKey = [redacted]v16VvbRbLLi1scAJ2Zz8J6ylKbrnLMNCUEVVc=
ListenPort = 51820
PostUp = iptables -t nat -A POSTROUTING -s 10.0.0.0/24 -o
eth- -j MASQUERADE
PostDown = iptables -t nat -D POSTROUTING -s 10.0.0.0/24 -
o eth- -j MASQUERADE

# Managing sysctl variable is preferably done outside of
wireguard though
PostUp = sysctl -w net.ipv4.ip_forward=1

[Peer]
...
```

client

```
[Interface]
Address = 10.0.0.2/24
ListenPort = 51820
PrivateKey = 4EcUDsTnuR0l[redacted]Yun3K7gCYPd5NALXJHXPaX7iVA=

[Peer]
PublicKey = CnmIgy[redacted]5nNJ1sP70BUkl+CBl/LXbQGKI048bHQE=
Endpoint = 108.51.[redacted]:51820
AllowedIPs = 0.0.0.0/0
```

Protokoll

Kryptographie ist vollständig festgelegt

Vorteile

- Einfachster Mechanismus zur Auswahl der Ciphersuites
- Keine Downgrade-Angriffe möglich
- Keine unsichere Fehlkonfiguration durch Admins/User

Nachteile

- Erfahrene User können Krypto nicht austauschen

Kryptographie (Parameter)

Schlüsselaushandlung

Diffie-Hellman auf elliptischen Kurven
(Curve25519)

Symmetrische Verschlüsselung

CHaCha20
(Schlüssellänge: 256 Bit)

Hashing/MAC

BLAKE2

Kryptographie (Anwendung)

- Nutzung statischer und kurzlebiger Schlüsselpaare zum Diffie-Hellman-Schlüsselaustausch
  → Perfect Forward Secrecy

- Keine direkte Übermittlung von statischen öffentlichen Schlüsseln
  → Verschleiert Identitäten der beteiligten Hosts für Außenstehende

- Optionaler symmetrischer Pre-Shared-Key Modus
  → Vorbeugung vor Quantenangriffen gegen asymmetrische Verfahren

Protokoll: Grundsätze (1)

- Basiert auf dem Noise-Framework (IK-Pattern)

- Alle Nachrichten sind authentifiziert:
  - Handshakes durch Public Keys der Sender
  - Payloads durch symmetrischen Sitzungs-Key aus Handshake

- Nicht-authentifizierte Nachrichten werden verworfen
  → DoS-Mitigation (vgl. TCP SYN-Flooding)
  → Keine Reaktion auf Port-Scans o. Ä.

Protokoll: Grundsätze (2)

- Session-Aufbau on-demand
  - 1 RTT Verzögerung für Anwendung bei erstem Paket
- Danach: Genau ein IP-Paket pro UDP-Datagramm
  - VPN-Übertragungen unterliegen grundsätzlich den Bedingungen des zugrundeliegenden Netzes (Paketverlust, Vertauschungen, usw.)
- Datagramme werden immer an die letzte bekannte IP-Adresse des Peers gesendet
- Regelmäßiges Durchwechseln (Rekeying) des symmetrischen Schlüssels

Protokoll: Überblick

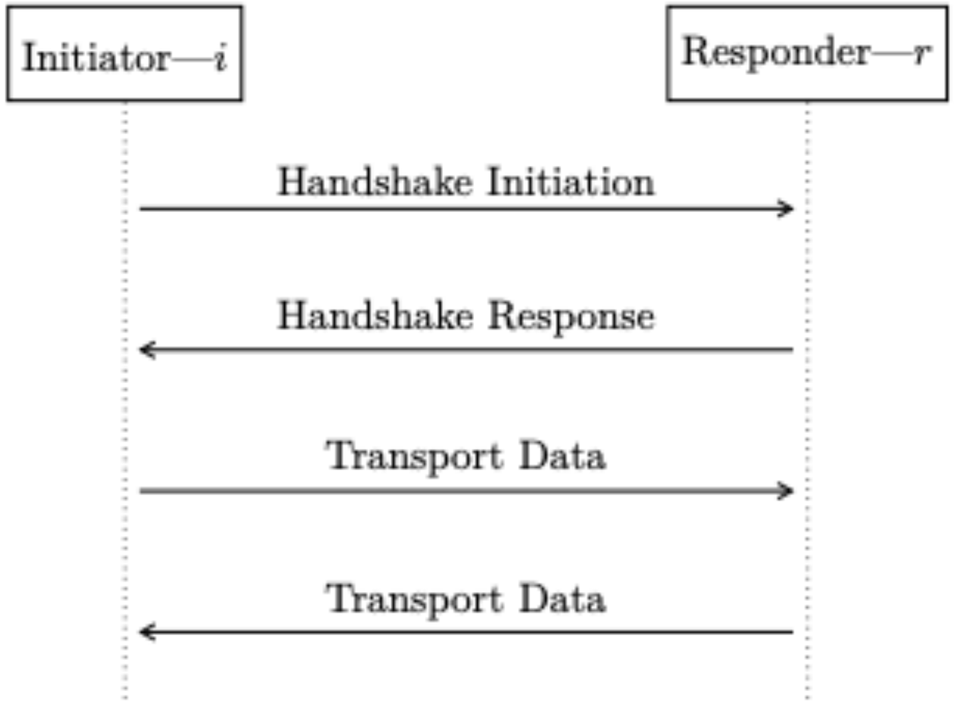
Initiator

Erster Peer, der Daten übertragen möchte und eine Ziel-IP kennt

Responder

Zweiter Peer

- Authentifizierter 2-Wege-Handshake etabliert symmetrischen Key
- Danach kann Payload (IP-Pakete) übertragen werden



[Whitepaper]

Protokoll: Handshake

- Herstellung eines gemeinsamen Zustandes durch 2 Nachrichten
  - Basiert auf Diffie-Hellman-Schlüsselaustausch
- Ableitung zweier symmetrischer Schlüssel aus diesem Zustand
  - Einer für jede Kommunikationsrichtung

Initiator → Responder

Type: 0x1 1 Byte	**Reserved: 0x000** 3 Byte
Sender ID 4 Byte	
Ephemeral PubKey 32 Byte	
Static 46 Byte (aus 32 Byte AEAD)	
Timestamp 28 Bytes (aus 12 Byte AEAD)	
mac1 16 Byte	**mac2** 16 Byte

Responder → Initiator

Type: 0x2 1 Byte	**Reserved: 0x000** 3 Byte
Sender ID 4 Byte	**Receiver ID** 4 Byte
Ephemeral PubKey 32 Byte	
Empty 16 Byte (aus 0 Byte AEAD)	
mac1 16 Byte	**mac2** 16 Byte

Protokoll: Daten

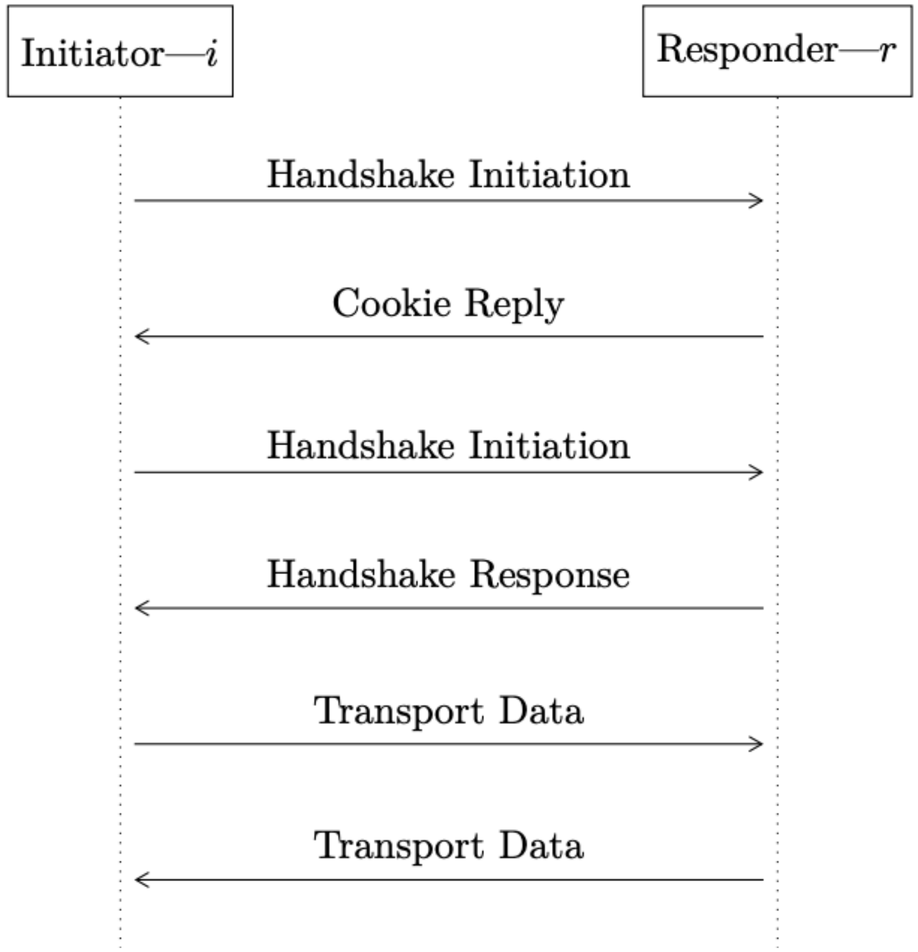
- Symmetrisch verschlüsselt
  → Vertraulichkeit & Authentizität
- Genau ein IP-Paket pro UDP-Datagramm

- Daten werden durch einen Counter vor Replay-Angriffen geschützt
  - Fenster mit akzeptierten vergangenen Countern

Type: 0x4 1 Byte	**Reserved**: **0x000** 3 Byte
Receiver ID 4 Byte	
Counter 8 Byte	
Paket 0 - Anz. Bytes symmetrisch verschlüsselt	

DoS Mitigation: Cookies im Handshake

- Optionaler Cookie-Mechanismus, falls Responder unter Last
- Cookie erfordert keine Zustandshaltung
- Sender muss Cookie dann in **`mac2`** verwenden



[Whitepaper]

Type: 0x3 1 Byte	**Reserved: 0x000** 3 Byte
Receiver ID 4 Byte	
Nonce 24 Byte	
Cookie 32 Byte (aus 16 Byte AEAD)	

Key-Rotation

- Neue Session mit neuen Keys etwa alle 2 Minuten
  - Initiator startet neuen Handshake
  - Keys der letzten Sitzung bleiben gültig für *in-transit* Pakete
- Responder hat nach Senden der Handshake-Nachricht keine Bestätigung vom Sender
  - Responder nutzt alte Keys, bis erstes Paket mit neuem Key empfangen

→ Insgesamt bis zu 3 Sessions im Speicher

Handshake Timeout, Keep Alive

Timeout für Handshakes

- 5 Sekunden
- Retry durch Initiator
- mit neuen Keys

Keep Alive

- Daten-Nachricht ohne Payload
- nach 10 Sekunden Funkstille
- je Richtung

Performance

Performance: Eigenschaften

- Linux: WireGuard läuft im Kernel-Space
  → Schneller als User-Space, wo VPN üblicherweise läuft

- Chiffre ist parallelisierbar
  → Kann alle Prozessorkerne gleichzeitig nutzen

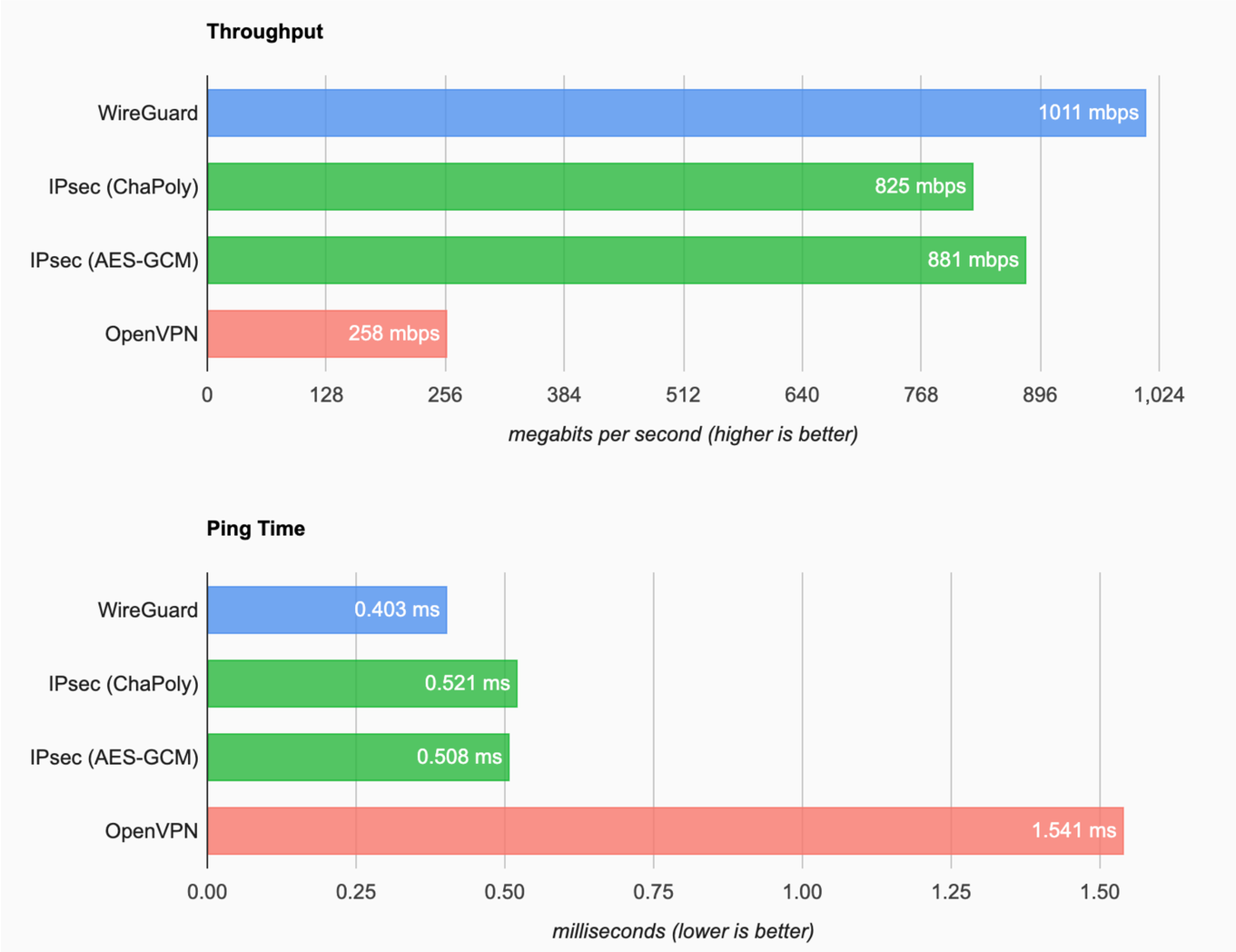
Performance: Vergleich

Setup

- 1Gbit/s Link
- **`iperf3`** über 30 Minuten
- OpenVPN in UDP mode
- IPsec mit je 1 Ciphersuite ähnlich zu OpenVPN / WireGuard

Ergebnis

- OpenVPN und IPsec laufen auf 100% CPU, WireGuard nicht
  - Limitiert Bandbreite



Administration

Administration: internes Netz

- Durch Zustandslosigkeit sehr stabil
- Einfache Lösung zum Verbinden mehrerer geschlossener Netze (z. B. zwischen verschiedenen Cloud-Hostern)

- Key-Exchange recht einfach bei Zugang zu allen Hosts

Administration: viele User

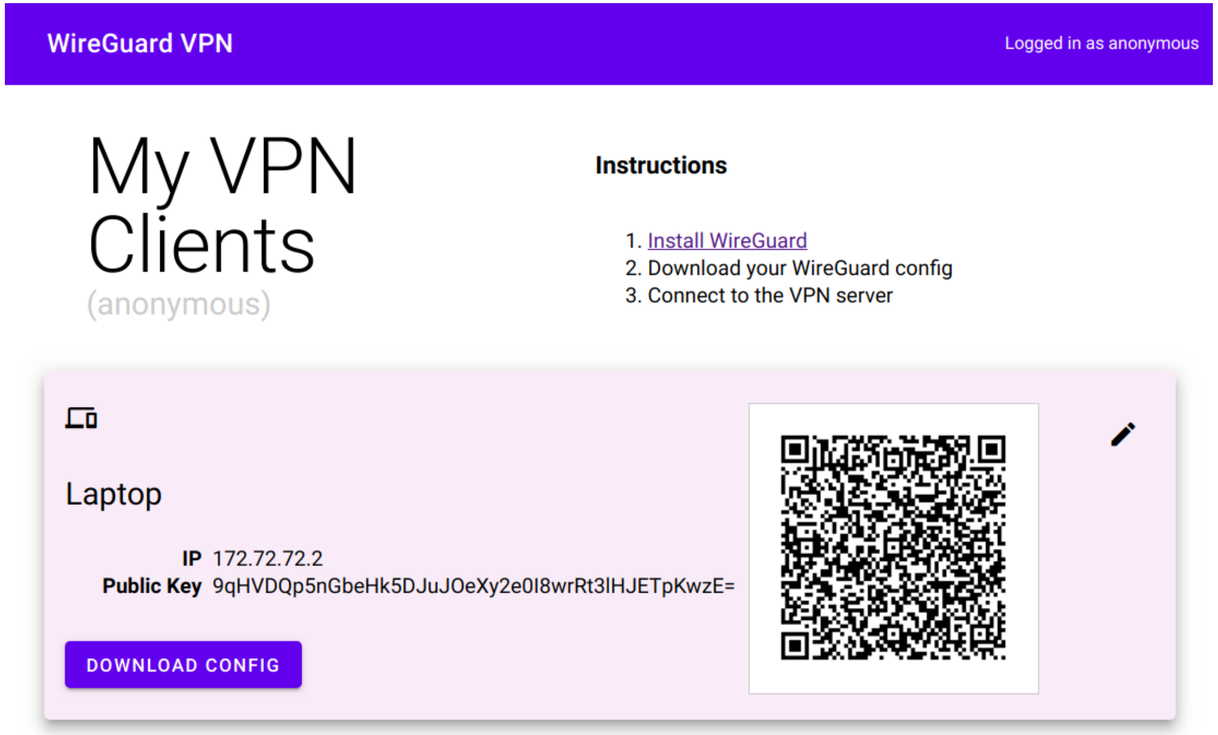
- z.B. Unternehmen, insbesondere auch nicht-technische User

Stolperfallen

- Private Key in Config
  → Config kann nicht ohne Private Key an User verteilt werden
- User verstehen Private/Public Keys nicht
  → Versehentliche Kompromittierung der Schlüssel
- Geräte-Authentifikation statt User-Authentifikation
  → User werden ihre Config auf mehreren Geräten nutzen wollen

Administration: Frontends

- `wg-ui`, `wireguard-ui`, Subspace
- Üblicherweise gesamte Client-Config **inkl. Private Key** serverseitig



[wg-ui]

Quellen

- **[Whitepaper]:** Donenfeld: Wireguard - Next Generation Kernel Network Tunnel (**`https://www.wireguard.com/papers/wireguard.pdf`**)
- **[wg-ui]: `https://github.com/EmbarkStudios/wg-ui/blob/main/wireguard-ui.png`**

Vielen Dank!

Kontakt:

✉ liz@lizbian.dev

@lizbian@chaos.social