ADMIN:DOCS

WireGuard VPN-Server auf Ubuntu einrichten: Kompletter Guide

WireGuard VPN auf Ubuntu einrichten: Praxis-Guide für Dual-Stack IPv4/IPv6, Unbound-DNS, UFW/nftables-Routing und automatisiertes Client-Setup per QR-Code.

Kategorie: Server environments **Veröffentlicht:** 26.08.2026 **Zuletzt geprüft:** 07.09.2026

WireGuard markiert einen grundlegenden Architekturwandel im Bereich virtueller privater Netzwerke (VPN). Im Gegensatz zu traditionellen Schwergewichten wie OpenVPN oder komplexen IPsec-Implementierungen arbeitet WireGuard direkt als schlankes **Linux-Kernel-Modul**. Mit weniger als 4.000 Zeilen Quellcode bietet das Protokoll eine minimale Angriffsfläche, moderne Kryptografie (Curve25519, ChaCha20-Poly1305, BLAKE2s) und hohe Datendurchsätze bei minimaler Latenz.

Der wesentliche Unterschied zu Protokollen wie OpenVPN oder IPsec/IKEv2 liegt in der Protokollarchitektur: WireGuard verzichtet auf komplexe Userspace-Daemons und mehrstufige Zertifikatsketten. Stattdessen führt WireGuard einen kryptografischen 1-RTT-Handshake (1 Round Trip Time) auf Basis des bewährten Noise-Protokollframeworks (`Noise_IKpsk2`) direkt im Linux-Kernel aus. Sitzungen werden geräuschlos ausgehandelt und bei IP-Wechseln (beispielsweise vom WLAN ins Mobilfunknetz) ohne Verbindungsabbruch weitergeführt (Roaming).

Dieser Leitfaden führt dich durch die vollständige Einrichtung eines produktionsreifen WireGuard-Gateways unter Ubuntu 24.04 LTS (Noble Numbat) und Ubuntu 26.04 LTS (Resolute Raccoon) inklusive sauber getrennter IPv4/IPv6-Architektur, konsistentem UFW-Routing, Pre-Shared Keys, Unbound-DNS und gehärtetem Client-Management per QR-Code.

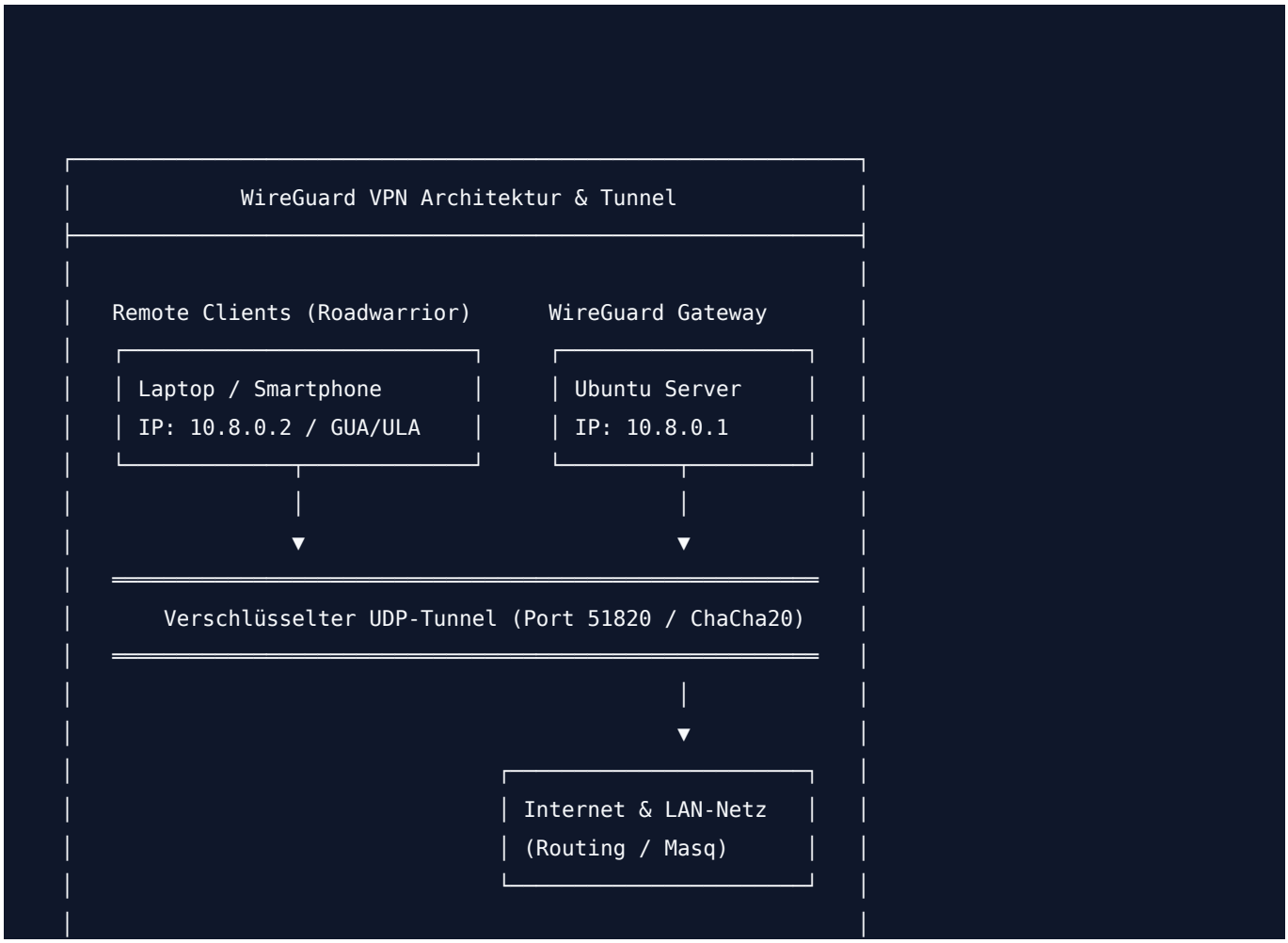
Die hier vermittelten Netzwerk- und Krypto-Grundlagen sind essenzieller Bestandteil moderner Serverumgebungen und orientieren sich an Standards von Prüfungen wie der LPIC-1 Serie.

Quelle: https://admindocs.de/de/serverumgebungen/ubuntu-24-04-server-einen-eigenen-wireguard-vpn-server-einrichten

> **Hinweis zur Kernel-Kompatibilität:** WireGuard ist seit Linux Kernel 5.6 fester Bestandteil des offiziellen Upstream-Kernels. Unter Ubuntu 24.04 und Ubuntu 26.04 sind keine externen Repositories (PPAs) oder Kernel-Module aus Fremdquellen mehr erforderlich.

WireGuard Architektur & Tunnel-Konzept

Im WireGuard-Modell sind alle Teilnehmer gleichberechtigte **Peers**. In der Server-Praxis dient der Ubuntu-Host als zentrales Gateway, das den Datenverkehr der Clients verschlüsselt annimmt und über die physische Netzwerkschnittstelle ins lokale Netz oder ins Internet weiterleitet:





IPv4- vs. IPv6-Architektur: Geroutetes Präfix vs. NAT66

Bei der Adressierung und dem Routing im VPN-Tunnel unterscheiden sich IPv4 und IPv6 grundlegend im empfohlenen Architekturansatz:

IPv4 (RFC 1918 & NAT/Masquerading):

Da öffentliche IPv4-Adressen knapp sind, erhalten VPN-Clients private Adressen (z. B. aus `10.8.0.0/24`). Der Server maskiert den ausgehenden Datenverkehr über seine öffentliche IP-Adresse mittels Source-NAT (Masquerading).

IPv6 – Bevorzugte Architektur (Geroutetes GUA-Präfix):

In IPv6 ist die direkte Ende-zu-Ende-Erreichbarkeit ohne NAT der Standard. Stellt dein Hostinganbieter dem Server ein geroutetes IPv6-Subnetz zur Verfügung (z. B. ein eigenes `/64`-Präfix aus einem `/56`- oder `/48`-Netz), erhalten die WireGuard-Clients echte Global Unicast Adressen (GUA). Der Server fungiert als reiner IPv6-Router ohne jegliches NAT.

IPv6 – Alternative Architektur (ULA mit NAT66):

Weist der Provider dem Host lediglich eine einzelne IPv6-Adresse oder ein direkt an der Schnittstelle liegendes `/64`-Netz zu, ohne ein separates Subnetz für Clients zu routen, ist natives Routing ohne Proxy-NDP nicht möglich. In diesem Fall kann ein Unique Local Address Präfix (ULA nach RFC 4193, z. B. `fd42:42:42::/64`) in Kombination mit NAT66 (IPv6-Masquerading) verwendet werden. NAT66 ist eine bewusste Übergangslösung und kein vollwertiges Äquivalent zu echter IPv6-Ende-zu-Ende-Konnektivität.

Die wichtigsten Eckdaten des Setups:

Parameter	Wert / Konfiguration	Zweck
VPN-Subnetz (IPv4)	`10.8.0.0/24`	Privater RFC1918-Adressraum für Clients (Server: `10.8.0.1`) mit NAT

Parameter	Wert / Konfiguration	Zweck
VPN-Subnetz (IPv6)	Geroutetes GUA (bevorzugt) oder ULA `fd42:42:42::/64`	IPv6-Clientadressen (Server: `...::1`); ULA erfordert NAT66
VPN-Port	`51820 / UDP`	Standard-Port für eingehenden WireGuard-Tunnelverkehr
Verschlüsselung	ChaCha20-Poly1305 (AEAD)	Authentifizierte symmetrische Verschlüsselung der Datenpakete
Schlüsselaustausch	Curve25519 (ECDH) + optionaler Pre-Shared Key (PSK)	1-RTT Noise IK Handshake; PSK ergänzt symmetrisches Schlüsselmaterial

> □□ **Hinweis zu Post-Quantum und Pre-Shared Keys (PSK):** WireGuard ist nicht von sich aus vollständig post-quanten-sicher. Der asymmetrische Schlüsselaustausch basiert auf Curve25519, was theoretisch durch zukünftige Quantencomputer mittels Shor-Algorithmus angreifbar wäre („Harvest Now, Decrypt Later“). Ein zusätzlicher Pre-Shared Key (`wg genpsk`) mischt 256 Bit symmetrisches Schlüsselmaterial in die Schlüsselableitung des Noise-Protokolls.
>
> Dies bietet eine wertvolle zusätzliche Verteidigungslinie gegen nachträgliche Entschlüsselung, ersetzt jedoch kein vollwertiges post-quanten-sicheres asymmetrisches KEM-Verfahren (wie ML-KEM).

Vorbereitung & Installation

1. System aktualisieren & Pakete installieren

Installiere das Paket `wireguard` (enthält das Kernelmodul und Steuerwerkzeuge), `qrencode` für mobile QR-Codes sowie Firewall-Tools:

```
sudo apt update && sudo apt upgrade -y
sudo apt install wireguard wireguard-tools qrencode iptables -y
```

> **iptables und nftables unter Ubuntu:** Unter Ubuntu 24.04 und 26.04 nutzt das System standardmäßig `iptables-nft`. Hierbei übersetzt ein Abstraktionslayer klassische `iptables`-Befehle direkt in das moderne `nftables`-Kernel-Subsystem. Werkzeuge wie UFW und bestehende Skripte arbeiten dadurch weiterhin zuverlässig und performant.

Überprüfe, ob das WireGuard-Kernelmodul geladen ist:

```
sudo modprobe wireguard
lsmod | grep wireguard
```

2. IP-Forwarding im Linux-Kernel aktivieren

Damit der Linux-Host Datenpakete zwischen dem VPN-Interface und dem physischen Netzwerkinterface weiterleiten darf, muss das IP-Forwarding für IPv4 und IPv6 aktiv sein. Erstelle eine dedizierte Konfigurationsdatei:

```
sudo tee /etc/sysctl.d/99-wireguard.conf << 'EOF'
# IPv4 Packet Forwarding aktivieren
net.ipv4.ip_forward = 1
```

```
# IPv6 Packet Forwarding aktivieren
net.ipv6.conf.all.forwarding = 1
EOF
```

Wende die Parameter sofort an:

```
sudo sysctl -p /etc/sysctl.d/99-wireguard.conf
```

Verifikation:

Prüfe, ob beide Parameter aktiv auf `1` stehen:

```
sysctl net.ipv4.ip_forward net.ipv6.conf.all.forwarding
```

Erwartete Ausgabe:

```
net.ipv4.ip_forward = 1
net.ipv6.conf.all.forwarding = 1
```

Schlüsselgenerierung (Server & Peer)

WireGuard nutzt asymmetrische Kryptografie: Jeder Teilnehmer besitzt einen privaten Schlüssel (geheim) und einen abgeleiteten öffentlichen Schlüssel (Public Key). Zusätzlich erzeugen wir für

jeden Client einen symmetrischen Pre-Shared Key (PSK).

Sichere das Konfigurationsverzeichnis mit strikten Zugriffsrechten ab (`0700`):

```
sudo mkdir -p /etc/wireguard
sudo chmod 700 /etc/wireguard
cd /etc/wireguard
```

Generiere das Schlüsselpaar für den Server mit sicherer Dateimaske (`umask 077`):

```
umask 077
wg genkey | tee server_private.key | wg pubkey > server_public.key
```

Lies den öffentlichen Schlüssel aus (dieser wird später in den Client-Profilen eingetragen):

```
echo "Server Public Key: $(cat server_public.key)"
```

Server-Konfiguration & Firewall-Architektur

Ermittle zunächst den Namen deiner primären Netzwerkschnittstelle mit Internetzugang (z. B. `eth0` , `enp1s0` oder `ens3`):

```
ip -o -4 route show to default | awk '{print $5}'
```

Im weiteren Verlauf verwenden wir stellvertretend das Interface `eth0`. Passe den Bezeichner bei Bedarf an die tatsächliche Netzwerkschnittstelle deines Servers an.

Um Sicherheitsrisiken und Regelkollisionen zu vermeiden, müssen Host-Firewall und Paketweiterleitung sauber aufeinander abgestimmt sein. Wir unterscheiden zwei Betriebsmodi:

Variante A (Empfohlen bei aktivem UFW):

UFW verwaltet Firewall, Forwarding und NAT zentral. Die Datei `wg0.conf` bleibt frei von `PostUp`/`PostDown`-Netfilter-Befehlen.

Variante B (Alternative ohne UFW):

Verwaltet ausschließlich die WireGuard-spezifischen Forwarding- und NAT-Regeln direkt über `PostUp` und `PostDown` in `wg-quick`. Host-INPUT-Regeln (wie Port 51820 UDP oder Port 53 für Unbound) müssen bei restriktiven Systemen separat in der Host-Firewall gepflegt werden.

> **Hinweis zu IPv6 in den Beispielen:** Die folgenden Konfigurationsbeispiele verwenden für IPv6 ein ULA-Präfix (`fd42:42:42::/64`), damit das Setup unabhängig von der Präfixdelegation des jeweiligen Hostingproviders direkt reproduzierbar bleibt. Steht dir ein für das WireGuard-Netz geroutetes GUA-Präfix zur Verfügung, ersetze die ULA-Adressen durch Adressen aus deinem gerouteten Präfix und lasse NAT66 in der Firewall vollständig weg.

Server-Konfigurationsdatei (`/etc/wireguard/wg0.conf`)

Erstelle die Datei `/etc/wireguard/wg0.conf`. Wenn du UFW verwendest (Variante A), benötigst du keine `PostUp`- und `PostDown`-Zeilen:

```
SERVER_PRIV_KEY=$(cat /etc/wireguard/server_private.key)

sudo tee /etc/wireguard/wg0.conf << EOF
[Interface]
# Server-Tunnel-IPs (Dual-Stack)
Address = 10.8.0.1/24, fd42:42:42::1/64
ListenPort = 51820
PrivateKey = ${SERVER_PRIV_KEY}

# Bei Variante A (UFW) entfallen PostUp/PostDown hier komplett.
# Peers werden nachfolgend registriert.
EOF
```

Schütze die Datei vor unberechtigtem Zugriff:

```
sudo chmod 600 /etc/wireguard/wg0.conf
```

Variante A (Empfohlen): Konsistente Firewall & NAT mit UFW

Das pauschale Setzen von `DEFAULT_FORWARD_POLICY="ACCEPT"` in `/etc/default/ufw` ist eine verbreitete, aber problematische Praxis, da es die Filterung für alle Schnittstellen des Hosts aushebelt.

Die saubere UFW-Architektur trennt Verantwortlichkeiten klar:

- **UFW CLI (`ufw route`):** Steuert Filterung und Forwarding gezielt auf Interface- und Subnetzebene.
- **`/etc/ufw/before.rules`:** Verwaltet das IPv4-Source-NAT (Masquerading), da UFW hierfür keine native CLI-Syntax bereitstellt.

- **`DEFAULT_FORWARD_POLICY="DROP"`:** Bleibt in `/etc/default/ufw` unverändert aktiv, um den Host abzusichern.

Schritt 1: IPv4-NAT in `/etc/ufw/before.rules` konfigurieren

Öffne `/etc/ufw/before.rules`:

```
sudo nano /etc/ufw/before.rules
```

Füge ganz am Anfang der Datei (noch **vor** dem ersten *`filter`*-Block) den folgenden `nat`-Block ein:

```
# NAT-Regeln für WireGuard VPN
*nat
:POSTROUTING ACCEPT [0:0]
-A POSTROUTING -s 10.8.0.0/24 -o eth0 -j MASQUERADE
COMMIT
```

Falls du für **IPv6** das ULA-Setup mit NAT66 verwendest, trage in:

```
/etc/ufw/before6.rules
```

analog vor `*filter` den Block:

```
*nat :POSTROUTING ... -A POSTROUTING -s fd42:42:42::/64 -o eth0 -j MASQUERADE COMMIT
```

Bei einem gerouteten GUA-Präfix entfällt dieser Schritt vollständig.

Schritt 2: UFW-Routing-Regeln und Portfreigabe einrichten

Konfiguriere das Forwarding direkt über die dafür vorgesehene UFW-Routing-Syntax:

```
# WireGuard VPN-Port für eingehenden Tunnelverkehr erlauben
sudo ufw allow 51820/udp comment 'WireGuard VPN Port'

# Lokalen DNS-Zugriff (Unbound) für WireGuard-Clients auf dem Host erlauben (UDP & TCP)
sudo ufw allow in on wg0 to any port 53 proto udp comment 'DNS from WireGuard'
sudo ufw allow in on wg0 to any port 53 proto tcp comment 'DNS from WireGuard'

# Gezieltes Forwarding: Datenverkehr von wg0 über das WAN-Interface (eth0) erlauben
sudo ufw route allow in on wg0 out on eth0 from 10.8.0.0/24

# Bei IPv6 (ULA oder geroutetes GUA):
sudo ufw route allow in on wg0 out on eth0 from fd42:42:42::/64
```

Der notwendige Rückverkehr für bestehende Verbindungen wird von UFWs Standardregelwerk (`-m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT` in `before.rules`) automatisch abgewickelt.

Lade UFW anschließend neu:

```
sudo ufw reload
```

Verifikation von UFW & Netfilter-Regeln:

```
# Zeigt Status, Portfreigaben und UFW-Routing-Regeln an
sudo ufw status verbose
```

Beachte: `ufw status` listet ausschließlich die über die UFW-CLI verwalteten Filter- und Routing-Regeln auf. Die direkt in `/etc/ufw/before.rules` und `/etc/ufw/before6.rules` definierten Low-Level-Netfilter-Regeln (wie das Masquerading) werden dort nicht aufgeführt. Den vollständigen Rohtabellen-Zustand kannst du mit `sudo ufw show raw` oder gezielt über die Kernel-Tabellen prüfen:

```
# IPv4-Masquerading prüfen
sudo iptables -t nat -S POSTROUTING

# Bei ULA-Setup mit NAT66: IPv6-Masquerading prüfen
sudo ip6tables -t nat -S POSTROUTING
```

Variante B (Alternative): Netfilter-Management der Tunnel-Regeln über `wg-quick`

Variante B verwaltet ausschließlich die für WireGuard erforderlichen Forwarding- und NAT-Regeln über `wg-quick`. Sie ersetzt keine vollständige Host-Firewall. Bei einer bestehenden restriktiven INPUT-Policy müssen der WireGuard-Port UDP/51820 auf dem WAN-Interface sowie – bei Verwendung von Unbound – UDP/TCP 53 auf `wg0` separat in der Host-Firewall freigegeben werden.

Um das Forwarding restriktiv zu halten, wird ausgehender Datenverkehr nur von `wg0` nach `eth0` erlaubt, während eingehender Datenverkehr auf `eth0 -> wg0` ausschließlich für bereits bestehende Verbindungen (`RELATED,ESTABLISHED`) akzeptiert wird:

```
PostUp = iptables -A FORWARD -i wg0 -o eth0 -j ACCEPT; iptables -A FORWARD -i eth0 -o wg0 -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT; iptables -t nat -A POSTROUTING -s 10.8.0.0/24 -o eth0 -j MASQUERADE
PostDown = iptables -D FORWARD -i wg0 -o eth0 -j ACCEPT; iptables -D FORWARD -i eth0 -o wg0 -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT; iptables -t nat -D POSTROUTING -s 10.8.0.0/24 -o eth0 -j MASQUERADE
```

Für das ULA-Setup kann analog eine Zeile mit `ip6tables` für `fd42:42:42::/64` ergänzt werden. Bei einem gerouteten GUA-Präfix entfällt die `-t nat`-Masquerading-Regel und es werden nur die `FORWARD`-Filterregeln benötigt.

WireGuard Service starten & aktivieren

Starte das Interface `wg0` über den Systemd-Dienst `wg-quick@wg0`:

```
sudo systemctl enable --now wg-quick@wg0
```

Technische Verifikation

Überprüfe systematisch den Status der Kernkomponenten:

Dienststatus prüfen:

```
sudo systemctl status wg-quick@wg0 --no-pager
```

Netzwerkschnittstelle und IP-Adressen verifizieren:

```
ip addr show wg0
```

Erwartung: Das Interface existiert, hat den Status `UP` und führt die Adressen `10.8.0.1/24` sowie `fd42:42:42::1/64`.

UDP-Socket kontrollieren:

```
sudo ss -ulnp | grep 51820
```

Erwartung: Der Kernel lauscht auf `0.0.0.0:51820` und `[::]:51820`.

WireGuard-Kernel-Status abfragen:

```
sudo wg show
```

Erwartete Ausgabe:

```
interface: wg0
```

```
  public key: <SERVER_PUBLIC_KEY>
  private key: (hidden)
  listening port: 51820
```

Client (Peer) anlegen & verbinden

Wir konfigurieren nun exemplarisch den ersten Client (z. B. Smartphone oder Notebook).

1. Client-Schlüssel und Pre-Shared Key generieren

Erstelle ein geschütztes Verzeichnis für Client-Profile:

```
sudo mkdir -p /etc/wireguard/clients
sudo chmod 700 /etc/wireguard/clients
cd /etc/wireguard/clients
umask 077

# Schlüsselpaar und symmetrischen PSK erzeugen
wg genkey | tee client1_private.key | wg pubkey > client1_public.key
wg genpsk > client1_preshared.key
```

2. Peer am Server registrieren

Trage den neuen Peer in `/etc/wireguard/wg0.conf` ein:

```
CLIENT1_PUB=$(cat client1_public.key)
CLIENT1_PSK=$(cat client1_preshared.key)
```

```
sudo tee -a /etc/wireguard/wg0.conf << EOF

# Client 1: Smartphone / Laptop
[Peer]
PublicKey = ${CLIENT1_PUB}
PresharedKey = ${CLIENT1_PSK}
AllowedIPs = 10.8.0.2/32, fd42:42:42::2/128
EOF
```

3. Konfiguration ohne Verbindungsunterbrechung neu laden (`wg syncconf`)

Verwende zum Übernehmen neuer Peers nicht `systemctl restart`, sondern den Befehl `wg syncconf`:

```
sudo wg syncconf wg0 <(wg-quick strip wg0)
```

Warum `wg syncconf`?

- Der Hilfsbefehl `wg-quick strip wg0` filtert Hilfsdirektiven (wie `Address`, `DNS` oder `PostUp`) heraus, sodass ein reines WireGuard-Regelwerk für das Kernel-Kommando `wg` übrig bleibt.
- `wg syncconf` vergleicht die gewünschte Konfiguration mit der aktuell im Kernel aktiven WireGuard-Konfiguration und wendet gezielt nur die notwendigen Änderungen (Delta) an.
- Bestehende Peer-Sessions und Verbindungen werden dadurch nicht unnötig gestört.
- Für reine Peer-Änderungen ist kein kompletter Neustart des Interfaces erforderlich, und Übertragungsstatistiken sowie Handshake-Timer bleiben erhalten.

4. Client-Konfigurationsdatei erstellen

Trage den erreichbaren Server-Endpunkt (öffentliche IPv4-Adresse oder FQDN) ein:

```
SERVER_ENDPOINT="vpn.example.com" # Oder öffentliche IP des Servers
SERVER_PUB=$(cat /etc/wireguard/server_public.key)
CLIENT1_PRIV=$(cat client1_private.key)

tee /etc/wireguard/clients/client1.conf << EOF
[Interface]
PrivateKey = ${CLIENT1_PRIV}
Address = 10.8.0.2/24, fd42:42:42::2/64
# DNS-Resolver des Gateways (Unbound in Schritt 7) oder vorab externe Resolver (z. B.
1.1.1.1)
DNS = 10.8.0.1, fd42:42:42::1

[Peer]
PublicKey = ${SERVER_PUB}
PresharedKey = ${CLIENT1_PSK}
Endpoint = ${SERVER_ENDPOINT}:51820
# 0.0.0.0/0, ::/0 leitet den gesamten Datenverkehr durch das VPN (Full-Tunnel)
AllowedIPs = 0.0.0.0/0, ::/0
# Hält die Verbindung hinter Stateful Firewalls / NAT-Routern stabil
PersistentKeepalive = 25
EOF
```

Full-Tunnel vs. Split-Tunneling:

Standardmäßig leitet `AllowedIPs = 0.0.0.0/0, ::/0` den gesamten Datenverkehr des Clients durch das VPN (Full-Tunnel). Soll hingegen nur der Zugriff auf das VPN-Subnetz oder interne Firmennetze über den Tunnel laufen, das reguläre Internetsurfen jedoch unberührt über die lokale Leitung des Clients verbleiben (Split-Tunneling), passe die Direktive im Client-Profil entsprechend an.

Beachte dabei die Konsistenz mit dem konfigurierten DNS-Server: Sollen neben den IPv4-Subnetzen auch der IPv6-Resolver (`fd42:42:42::1`) oder interne IPv6-Dienste über den Tunnel erreichbar sein, muss das entsprechende IPv6-VPN-Präfix ebenfalls in `AllowedIPs` aufgeführt werden:

```
# Dual-Stack Split-Tunnel (inkl. IPv6-DNS-Resolver)
AllowedIPs = 10.8.0.0/24, fd42:42:42::/64, 192.168.10.0/24
```

Wird stattdessen ein reiner IPv4-Split-Tunnel (`AllowedIPs = 10.8.0.0/24, 192.168.10.0/24`) eingerichtet, darf im Client-Profil unter `DNS =` ausschließlich die IPv4-Adresse des Resolvers (`DNS = 10.8.0.1`) eingetragen werden, da Anfragen an `fd42:42:42::1` andernfalls mangels Routing-Eintrag vom Client verworfen werden.

> **DNS-Eintrag vs. Transportverschlüsselung:** Der Parameter `DNS = ...` in der Client-Konfiguration weist dem Betriebssystem lediglich die IP-Adresse des zu verwendenden Resolvers zu. Dies aktiviert **kein** DNS-over-TLS (DoT) oder DNS-over-HTTPS (DoH). Bei einem Full-Tunnel werden DNS-Pakete zwar innerhalb des VPN-Tunnels verschlüsselt zum Server transportiert, verlassen diesen zum Upstream-Resolver jedoch standardmäßig unverschlüsselt über Port 53 (sofern kein lokaler DoT-Resolver wie Unbound auf dem Server läuft).

5. QR-Code für Smartphones erzeugen

Für Mobilgeräte mit der offiziellen WireGuard-App lässt sich die Konfiguration direkt im Terminal als ANSI-QR-Code anzeigen:

```
qrencode -t ansiutf8 < /etc/wireguard/clients/client1.conf
```

Öffne die App auf dem Smartphone, wähle **+** → **QR-Code scannen**, richte die Kamera auf das Terminal und aktiviere den Tunnel.

Automatisierung: Gehärtetes Client-Management-Skript

Um neue Clients sicher und reproduzierbar per Einzeiler anzulegen, empfiehlt sich ein gehärtetes Verwaltungsskript. Das folgende Skript vermeidet externe Webabfragen, validiert Eingaben strikt, verhindert Adresskollisionen, konvertiert IPv6-Suffixe korrekt hexadezimal und arbeitet über temporäre Dateien, File-Locking und einen kontrollierten Rollback-Mechanismus fehlertolerant.

Erstelle die Datei `/usr/local/bin/add-wireguard-client`:

```
sudo tee /usr/local/bin/add-wireguard-client << 'EOF'
#!/bin/bash
set -euo pipefail

# ==============================================================================
# Konfiguration
# ==============================================================================
WG_DIR="/etc/wireguard"
WG_CONF="$WG_DIR/wg0.conf"
CLIENT_DIR="$WG_DIR/clients"
LOCK_FILE="/var/lock/wireguard-client.lock"

# Trage hier den festen FQDN oder die öffentliche IP deines Servers ein
SERVER_ENDPOINT="vpn.example.com"
SERVER_PORT=51820
SERVER_PUB_KEY_FILE="$WG_DIR/server_public.key"

# Adressräume
VPN_NET_PREFIX="10.8.0"
VPN_IPV6_PREFIX="fd42:42:42::"
DNS_SERVERS="10.8.0.1, fd42:42:42::1"

# ==============================================================================
# Validierung & Concurrency-Locking
# ==============================================================================
if [ "$EUID" -ne 0 ]; then
    echo "Fehler: Dieses Skript muss mit administrativen Rechten (sudo) ausgeführt
werden." >&2
```

```
    exit 1
fi

if [ -z "${1:-}" ]; then
    echo "Verwendung: sudo $0 <client-name> [ip-suffix]" >&2
    echo "Beispiel:   sudo $0 notebook 3" >&2
    exit 1
fi

CLIENT_NAME="$1"
IP_SUFFIX="${2:-}"

# Client-Namen validieren (2-32 Zeichen, alphanumerisch sowie '-' und '_')
if ! [[ "$CLIENT_NAME" =~ ^[a-zA-Z0-9_-]{2,32}$ ]]; then
    echo "Fehler: Ungültiger Client-Name '$CLIENT_NAME'. Erlaubt sind 2-32 Zeichen [a-zA-
Z0-9_-]." >&2
    exit 1
fi

# Concurrency-Locking per flock gegen Race Conditions
exec 200>"$LOCK_FILE"
if ! flock -n 200; then
    echo "Fehler: Eine andere Instanz dieses Skripts läuft bereits." >&2
    exit 1
fi

if [ ! -f "$WG_CONF" ] || [ ! -f "$SERVER_PUB_KEY_FILE" ]; then
    echo "Fehler: $WG_CONF oder $SERVER_PUB_KEY_FILE existiert nicht." >&2
    exit 1
fi

CONF_FILE="$CLIENT_DIR/${CLIENT_NAME}.conf"
if [ -f "$CONF_FILE" ]; then
    echo "Fehler: Konfigurationsdatei '$CONF_FILE' existiert bereits." >&2
    exit 1
fi

if grep -qE "^\s*#\s*Peer:\s*${CLIENT_NAME}\s*$" "$WG_CONF"; then
    echo "Fehler: Ein Peer namens '$CLIENT_NAME' ist bereits in $WG_CONF hinterlegt." >&2
    exit 1
fi

# ================================================================================
# IP-Vergabe & Kollisionsprüfung (IPv4 dezimal, IPv6 hexadezimal)
```

```
# ================================================================================
if [ -n "$IP_SUFFIX" ]; then
    if ! [[ "$IP_SUFFIX" =~ ^[0-9]+$ ]] || [ "$IP_SUFFIX" -lt 2 ] || [ "$IP_SUFFIX" -gt
254 ]; then
        echo "Fehler: Das IP-Suffix muss eine Ganzzahl zwischen 2 und 254 sein." >&2
        exit 1
    fi
    printf -v IPV6_HEX '%x' "$IP_SUFFIX"
    CANDIDATE_IPV4="${VPN_NET_PREFIX}.${IP_SUFFIX}"
    CANDIDATE_IPV6="${VPN_IPV6_PREFIX}${IPV6_HEX}"

    if grep -qF "$CANDIDATE_IPV4" "$WG_CONF" || grep -qF "$CANDIDATE_IPV6" "$WG_CONF";
then
        echo "Fehler: IP-Adresse ($CANDIDATE_IPV4 oder $CANDIDATE_IPV6) ist in $WG_CONF
bereits vergeben." >&2
        exit 1
    fi
else
    # Automatische Zuweisung der nächsten freien IP
    IP_SUFFIX=0
    for i in $(seq 2 254); do
        TEST_IPV4="${VPN_NET_PREFIX}.${i}"
        printf -v TEST_HEX '%x' "$i"
        TEST_IPV6="${VPN_IPV6_PREFIX}${TEST_HEX}"
        if ! grep -qF "$TEST_IPV4" "$WG_CONF" && ! grep -qF "$TEST_IPV6" "$WG_CONF"; then
            IP_SUFFIX="$i"
            break
        fi
    done
    if [ "$IP_SUFFIX" -eq 0 ]; then
        echo "Fehler: Keine freien Adressen im Bereich ${VPN_NET_PREFIX}.2-254 verfügbar."
>&2
        exit 1
    fi
fi

CLIENT_IPV4="${VPN_NET_PREFIX}.${IP_SUFFIX}"
printf -v IPV6_HEX '%x' "$IP_SUFFIX"
CLIENT_IPV6="${VPN_IPV6_PREFIX}${IPV6_HEX}"

# ================================================================================
# Sichere Erstellung über temporäre Dateien mit Rollback
# ================================================================================
mkdir -p "$CLIENT_DIR"
```

```
    umask 077

    TMP_CLIENT_CONF=$(mktemp "$CLIENT_DIR/.client.XXXXXX")
    TMP_WG_CONF=$(mktemp --suffix=.conf "$WG_DIR/wgtmp.XXXXXX")
    chmod 600 "$TMP_CLIENT_CONF" "$TMP_WG_CONF"

    cleanup() {
        rm -f "$TMP_CLIENT_CONF" "$TMP_WG_CONF" "${WG_CONF}.bak"
    }
    trap cleanup EXIT INT TERM

    echo "==> Generiere Schlüsselmaterial für Peer: $CLIENT_NAME ($CLIENT_IPV4 /
    $CLIENT_IPV6)..."
    PRIV_KEY=$(wg genkey)
    PUB_KEY=$(echo "$PRIV_KEY" | wg pubkey)
    PSK=$(wg genpsk)
    SERVER_PUB=$(cat "$SERVER_PUB_KEY_FILE")

    # Client-Profil temporär schreiben
    cat > "$TMP_CLIENT_CONF" << CLIENT_CONF
    [Interface]
    PrivateKey = ${PRIV_KEY}
    Address = ${CLIENT_IPV4}/24, ${CLIENT_IPV6}/64
    DNS = ${DNS_SERVERS}

    [Peer]
    PublicKey = ${SERVER_PUB}
    PresharedKey = ${PSK}
    Endpoint = ${SERVER_ENDPOINT}:${SERVER_PORT}
    AllowedIPs = 0.0.0.0/0, ::/0
    PersistentKeepalive = 25
    CLIENT_CONF

    # Neue Server-Konfiguration vorbereiten
    cp -p "$WG_CONF" "$TMP_WG_CONF"
    cat >> "$TMP_WG_CONF" << PEER_CONF

    # Peer: ${CLIENT_NAME}
    [Peer]
    PublicKey = ${PUB_KEY}
    PresharedKey = ${PSK}
    AllowedIPs = ${CLIENT_IPV4}/32, ${CLIENT_IPV6}/128
    PEER_CONF
```

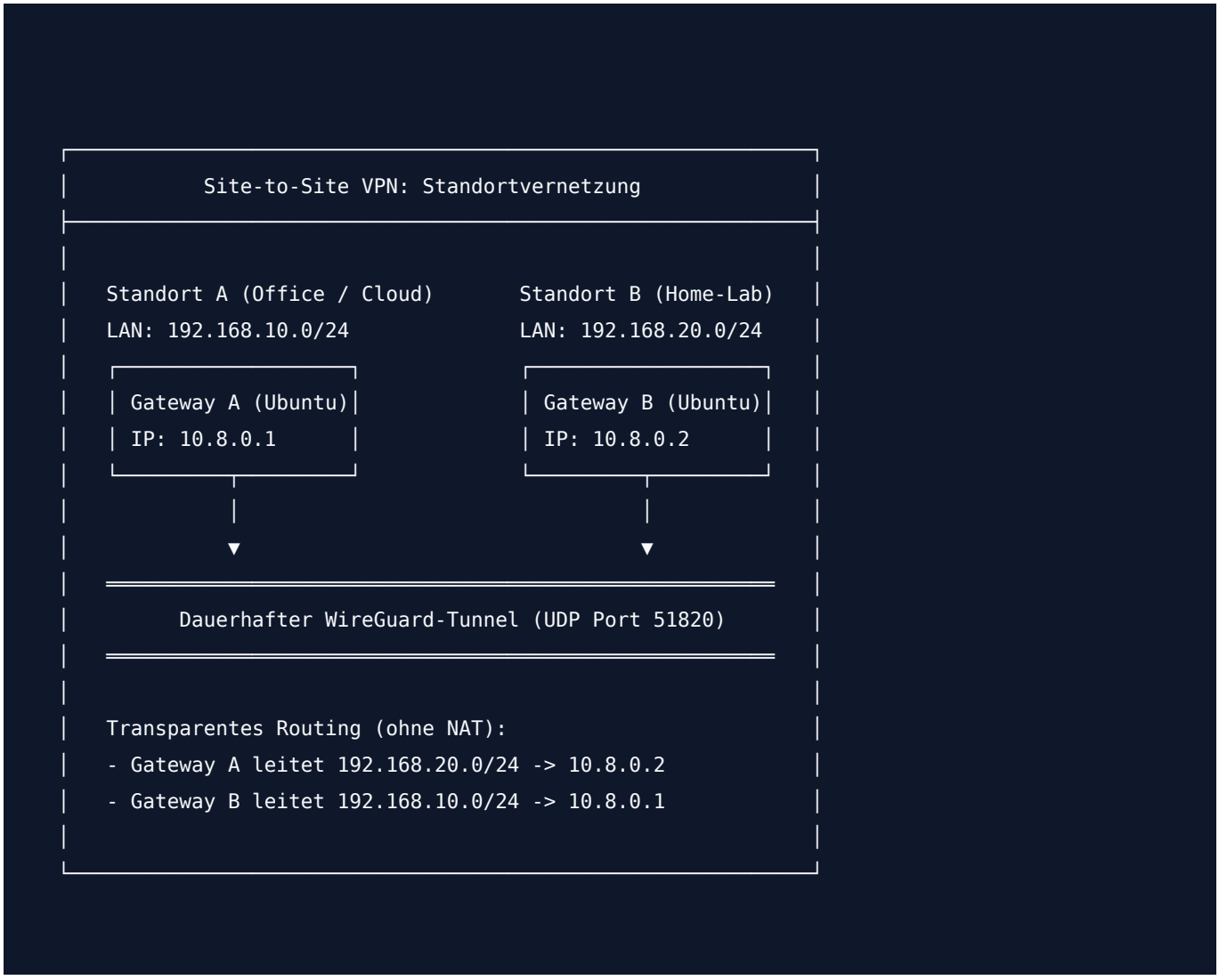
```
    # Syntaxprüfung der neuen Konfiguration vor Aktivierung
    if ! wg-quick strip "$TMP_WG_CONF" > /dev/null 2>&1; then
        echo "Fehler: Die generierte Server-Konfiguration ist syntaktisch fehlerhaft.
    Abbruch." >&2
        exit 1
    fi

    # Server-Konfiguration mit Backup aktualisieren
    cp -p "$WG_CONF" "${WG_CONF}.bak"
    cp "$TMP_WG_CONF" "$WG_CONF"

    # Live-Abgleich mit dem Kernel durchführen
    if ! wg syncconf wg0 <(wg-quick strip "$WG_CONF"); then
        echo "Fehler: Kernel-Abgleich via wg syncconf fehlgeschlagen. Führe Rollback durch..."
    >&2
        cp -p "${WG_CONF}.bak" "$WG_CONF"
        if ! wg syncconf wg0 <(wg-quick strip "$WG_CONF"); then
            echo "KRITISCH: Rollback der laufenden WireGuard-Konfiguration fehlgeschlagen."
    >&2
        fi
        exit 1
    fi

    # Client-Profil atomar an Zielort verschieben
    mv "$TMP_CLIENT_CONF" "$CONF_FILE"
    rm -f "${WG_CONF}.bak"

    echo "==> Profil erfolgreich gespeichert: $CONF_FILE"
    echo ""
    echo "=== QR-CODE FÜR MOBILGERÄTE ==="
    qrencode -t ansiutf8 < "$CONF_FILE"
    EOF

    sudo chmod +x /usr/local/bin/add-wireguard-client
```

Nun können neue Clients sicher erzeugt werden:

```
sudo add-wireguard-client tablet
sudo add-wireguard-client macbook 5
```

Site-to-Site VPN: Standortvernetzung zweier Netzwerke

Während das Roadwarrior-Setup einzelne Endgeräte anbindet, koppelt ein **Site-to-Site VPN** zwei komplette Netzwerke transparent miteinander. Geräte an Standort A (`192.168.10.0/24`) können direkt ohne Clientsoftware auf Server an Standort B (`192.168.20.0/24`) zugreifen:



Wichtige Voraussetzungen für Site-to-Site

Keine überlappenden IP-Subnetze:

Die beiden Standorte müssen zwingend unterschiedliche IP-Bereiche nutzen (z. B. `192.168.10.0/24` und `192.168.20.0/24`). Nutzen beide Seiten denselben Adressbereich (wie das gängige `192.168.1.0/24`), ist direktes Routing unmöglich.

IP-Forwarding auf beiden Gateways:

Auf Gateway A und Gateway B muss `net.ipv4.ip_forward = 1` aktiv sein.

Kein NAT zwischen den Standorten: Im Tunnelverkehr zwischen den Netzen wird typischerweise kein Masquerading eingesetzt, damit Quell-IPs für Monitoring, Logging und Firewall-Regeln unverändert erhalten bleiben.

Rückrouten im LAN-Router: Clients in Standort A senden Pakete an ihr Standard-Gateway (z. B. FRITZ!Box oder pfSense). Der Hauptrouter muss wissen, wie das entfernte Netz erreichbar ist. Daher muss im Hauptrouter von Standort A eine statische Route hinterlegt werden:

Zielnetz:

`192.168.20.0/24`

Gateway / Next Hop:

- IP-Adresse von Gateway A im lokalen Netz (z. B. `192.168.10.254`).
- Analog auf Standort B: `192.168.10.0/24` via `192.168.20.254`.

1. Konfiguration Gateway A (`/etc/wireguard/wg0.conf`)

Gateway A fungiert als Responder mit öffentlich erreichbarer Adresse oder DynDNS:

```
[Interface]
Address = 10.8.0.1/24
```

```
ListenPort = 51820
PrivateKey = <GATEWAY_A_PRIVATE_KEY>

# Forwarding zwischen lokalem LAN (eth0) und VPN (wg0) erlauben (ohne NAT)
PostUp = iptables -A FORWARD -i wg0 -o eth0 -j ACCEPT; iptables -A FORWARD -i eth0 -o wg0
-j ACCEPT
PostDown = iptables -D FORWARD -i wg0 -o eth0 -j ACCEPT; iptables -D FORWARD -i eth0 -o
wg0 -j ACCEPT

# Peer: Gateway B
[Peer]
PublicKey = <GATEWAY_B_PUBLIC_KEY>
PresharedKey = <SITE_TO_SITE_PSK>
# Erlaube sowohl die Tunnel-IP als auch das gesamte Remote-LAN
AllowedIPs = 10.8.0.2/32, 192.168.20.0/24
```

2. Konfiguration Gateway B (`/etc/wireguard/wg0.conf`)

Gateway B baut die Verbindung aktiv zu Gateway A auf:

```
[Interface]
Address = 10.8.0.2/24
PrivateKey = <GATEWAY_B_PRIVATE_KEY>

PostUp = iptables -A FORWARD -i wg0 -o eth0 -j ACCEPT; iptables -A FORWARD -i eth0 -o wg0
-j ACCEPT
PostDown = iptables -D FORWARD -i wg0 -o eth0 -j ACCEPT; iptables -D FORWARD -i eth0 -o
wg0 -j ACCEPT

# Peer: Gateway A
[Peer]
PublicKey = <GATEWAY_A_PUBLIC_KEY>
PresharedKey = <SITE_TO_SITE_PSK>
Endpoint = gateway-a.example.com:51820
AllowedIPs = 10.8.0.1/32, 192.168.10.0/24
PersistentKeepalive = 25
```

3. Bidirektionale Verifikation

Überprüfe die Verbindung in beiden Richtungen:

```
# 1. Tunnel-Handshake prüfen (auf beiden Gateways)
sudo wg show

# 2. Ping auf die entfernte Tunnel-IP (von Gateway A)
ping -c 3 10.8.0.2

# 3. Ping auf einen Host im Remote-LAN (von Gateway A)
ping -c 3 192.168.20.50

# 4. Traceroute von einem regulären Client im LAN A
traceroute 192.168.20.50
```

Bei ungefiltertem Traceroute-/ICMP-Verkehr sollte typischerweise zunächst das lokale WireGuard-Gateway und anschließend das entfernte Gateway sichtbar werden. Fehlende Zwischenstationen (z. B. Sternchen `*`) bedeuten dabei nicht automatisch einen Routing-Fehler, sondern treten häufig auf, wenn Firewalls oder Router zwischen den Standorten ICMP-Time-Exceeded-Nachrichten verwerfen.

Eigener DNS-Server & Interne Namensauflösung (Unbound)

Sollen VPN-Clients interne Hostnamen (z. B. `nas.intern` oder `wiki.corp`) auflösen können, ohne DNS-Anfragen unverschlüsselt an öffentliche Resolver weiterzuleiten, bietet sich ein lokaler Unbound-DNS-Resolver auf dem Server an.

Installiere Unbound:

```
sudo apt install unbound -y
```

Unbound-Konfiguration mit DNS-over-TLS (DoT)

Wir unterscheiden hierbei sauber zwischen lokaler Namensauflösung und der Art der Weiterleitung:

- **Variante 1 (Empfohlen): Weiterleitung über DNS-over-TLS (DoT):** Verschlüsselt externe DNS-Anfragen zwischen deinem Server und Upstream-Resolvern (z. B. Quad9) über Port 853 mit Zertifikatsvalidierung.
- **Variante 2: Standard-Weiterleitung:** Leitet Anfragen unverschlüsselt über den traditionellen UDP/TCP-Port 53 weiter.

Erstelle die Konfigurationsdatei `/etc/unbound/unbound.conf.d/wireguard.conf`:

```
sudo tee /etc/unbound/unbound.conf.d/wireguard.conf << 'EOF'
server:
    interface: 10.8.0.1
    interface: fd42:42:42::1
    interface: 127.0.0.1
    interface: ::1
    access-control: 10.8.0.0/24 allow
    access-control: fd42:42:42::/64 allow
    access-control: 127.0.0.0/8 allow
    access-control: ::1 allow

    # Sicherheits- & Härtungseinstellungen
    hide-identity: yes
    hide-version: yes
    harden-glue: yes
    harden-dnssec-stripped: yes
```

```
    tls-cert-bundle: "/etc/ssl/certs/ca-certificates.crt"

    # Lokale Infrastrukturzone definieren
    local-zone: "intern." static
    local-data: "nas.intern. IN A 192.168.10.50"
    local-data: "wiki.intern. IN A 192.168.10.51"
    local-data: "router.intern. IN A 10.8.0.1"

# Variante 1: Verschlüsselte Upstream-Weiterleitung per DNS-over-TLS (Port 853)
forward-zone:
    name: "."
    forward-tls-upstream: yes
    forward-addr: 9.9.9.9@853#dns.quad9.net
    forward-addr: 149.112.112.112@853#dns.quad9.net

# Variante 2 (Alternative für Standard-DNS ohne TLS auf Port 53):
# forward-zone:
#     name: "."
#     forward-tls-upstream: no
#     forward-addr: 9.9.9.9
#     forward-addr: 149.112.112.112
EOF
```

Starte Unbound und prüfe den Dienst:

```
sudo systemctl enable --now unbound
sudo systemctl status unbound --no-pager
```

Trage in den Client-Profilen unter `[Interface]` nun `DNS = 10.8.0.1, fd42:42:42::1, intern` ein. Der Client richtet alle DNS-Anfragen durch den VPN-Tunnel an Unbound. Bei Verwendung eines gerouteten GUA-Präfixes müssen die ULA-Adressen durch deine GUA-Serveradressen ersetzt werden. Achte darauf, dass die Host-Firewall (UFW oder nftables) Anfragen auf Port 53 über das Interface `wg0` zulässt (wie in den Firewall-Abschnitten eingerichtet), da DNS-Verkehr an das Gateway als lokaler Host-Traffic (INPUT) gefiltert wird.

> ◫ **DNS-Integration von `wg-quick` unter Linux:** `wg-quick` verarbeitet die Direktive `DNS =` über das Kommando `resolvconf`. Unter Ubuntu kann diese Schnittstelle über die resolvconf-Kompatibilität von `systemd-resolved` bereitgestellt werden. Wird ein WireGuard-Profil dagegen über NetworkManager verwaltet oder importiert, übernimmt NetworkManager die DNS-Integration direkt.
>
> Prüfe am Client nach dem Verbindungsaufbau mit `resolvectl status wg0`, ob die zugewiesenen Resolver und – abhängig von der verwendeten Resolver-Integration – die konfigurierte Such-/Routing-Domain für `intern` wirksam sind.

Strikter VPN-Kill-Switch für mobile Clients

Auf mobilen Laptops (etwa in öffentlichen Netzwerken oder Hotels) soll bei einem Verbindungsabbruch des VPN-Tunnels verhindert werden, dass unverschlüsselter Datenverkehr über das lokale Interface fließt.

> ⚠ **Achtung vor Aktivierung des Kill-Switches:** Der Befehl `sudo ufw default deny outgoing` sperrt sofort jeglichen ausgehenden Netzwerkverkehr des Rechners. Führst du diesen Schritt unbedacht über eine bestehende SSH-Verbindung aus oder fehlen notwendige Ausnahmen für DHCP und LAN, schneidest du dich unmittelbar vom System ab. Richte die Ausnahmen exakt in der angegebenen Reihenfolge ein.

Architektur & bewusste Designentscheidung: IPv4-Bootstrap & strikter Tunnel

Ein mobiler Linux-Client (Laptop) bewegt sich häufig in wechselnden Gast-WLANs, Hotels oder mobilen Hotspots. Ein völlig universeller Dual-Stack-Kill-Switch für beliebige fremde Netze ist auf Betriebssystemebene komplex, da sich lokale IPv6-Präfixe, Router Advertisements und DNS-Verhalten dynamisch ändern.

Wir setzen daher auf eine **bewusste, gehärtete Referenzarchitektur**:

Feste IPv4-Serveradresse für den Handshake:

Der Client baut den WireGuard-Tunnel gezielt über die öffentliche IPv4-Adresse des Servers auf (`Endpoint = <SERVER_IPV4>:51820`). Dadurch entfällt vor dem Tunnelaufbau jegliche unverschlüsselte DNS-Bootstrap-Kommunikation nach außen.

Minimaler IPv4-Bootstrap über das physische Interface:

Für den Verbindungsaufbau werden Loopback, DHCPv4 (UDP-Ports 67/68) und der WireGuard-Handshake (UDP-Port 51820) auf der physischen Schnittstelle freigegeben. Neben Loopback, DHCPv4 und dem WireGuard-Handshake kann **optional** der Zugriff auf das unmittelbar angeschlossene lokale Netz erlaubt werden. Diese Ausnahme bedeutet bewusst, dass lokaler LAN-Verkehr nicht durch den VPN-Tunnel läuft; in fremden Hotel- oder Gast-WLANs sollte dieser Schritt zur maximalen Abschottung weggelassen werden.

Sämtlicher Nutzdatenverkehr (IPv4 & IPv6) ausschließlich über `wg0`:

Sobald der Tunnel steht, fließt der gesamte reguläre Datenverkehr – IPv4 wie auch IPv6 – verschlüsselt durch das Interface `wg0`.

IPv6-Nutzdaten außerhalb von `wg0` blockiert:

Regulärer ausgehender IPv6-Nutzdatenverkehr über die physische Schnittstelle wird durch `default deny outgoing` vollständig unterbunden (kein IPv6-Leak am Tunnel vorbei).

Lokale Link-Kommunikation:

Die für die grundlegende lokale Schnittstelleninitialisierung erforderlichen ICMPv6-Pakete (Neighbor Discovery, Router Advertisements) werden in UFWs systemweiten Standardregeln (`/etc/ufw/before6.rules`) auf Link-Local-Ebene (`fe80::/10`) automatisch verarbeitet, sodass das physische Interface im lokalen Netz stabil bleibt, ohne Internet-Traffic durchzulassen.

UFW-Regeln für den Kill-Switch (am Linux-Client)

Führe die Konfiguration auf dem Client schrittweise durch:

```
# 1. Lokales Loopback-Interface vollständig erlauben
sudo ufw allow out on lo to any

# 2. DHCPv4 für die lokale IP-Aushandlung am physischen Interface erlauben
sudo ufw allow out 67:68/udp comment 'DHCP Client'

# 3. Optional: Lokales Subnetz erlauben (nur bei Bedarf, z. B. Heimnetz; in fremden Netzen weglassen)
sudo ufw allow out to 192.168.1.0/24 comment 'Lokales LAN (Optional)'

# 4. Verbindungsaufbau zum WireGuard-Server über IPv4 erlauben (Feste Server-IP!)
sudo ufw allow out to <SERVER_IPV4> proto udp port 51820 comment 'WireGuard Handshake'

# 5. Gesamten Nutzdatenverkehr (IPv4 und IPv6) über das Tunnel-Interface erlauben
sudo ufw allow out on wg0 to any

# 6. Eingehenden und ausgehenden Datenverkehr auf allen Interfaces standardmäßig blockieren
sudo ufw default deny incoming
sudo ufw default deny outgoing

# 7. Firewall aktivieren
sudo ufw enable
```

Deaktivierung im Notfall:

Falls du den Kill-Switch wieder aufheben möchtest:

```
sudo ufw default allow outgoing
sudo ufw reload
```

Monitoring & Fehlerbehebung

Eine erfolgreiche Fehlersuche unterscheidet strikt zwischen drei Ebenen: **Kryptografischer Handshake**, **IP-Routing** und **DNS-Auflösung**.

1. Handshake-Prüfung (`wg show`)

Führe auf dem Server oder Client folgenden Befehl aus:

```
sudo wg show
```

Beispielausgabe:

```
interface: wg0
  public key: 4HkL...8Xg=
  private key: (hidden)
  listening port: 51820

peer: 4HkL...8Xg=
  preshared key: (hidden)
  endpoint: 203.0.113.42:54120
  allowed ips: 10.8.0.2/32, fd42:42:42::2/128
  latest handshake: 14 seconds ago
  transfer: 2.14 MiB received, 18.76 MiB sent
```

> ⚠ **Handshake fehlt bei Verbindungsversuchen?** WireGuard erzeugt Handshakes bedarfsgesteuert: Ohne ausgehenden oder eingehenden Datenverkehr findet kein Rekeying statt; ein älterer `latest handshake` ist im Ruhezustand daher völlig normal. Wenn

> jedoch bei aktiv erzeugtem Traffic (z. B. einem laufenden Ping) kein aktueller Handshake zustande kommt oder `latest handshake` trotz wiederholter Verbindungsversuche nicht aktualisiert wird, liegen meist folgende Ursachen vor:
>
> - Der UDP-Port `51820` ist in einer Firewall (z. B. Cloud-Security-Group, Router) blockiert.
> - Der Server-Endpunkt (Domain oder IPv4) ist für den Client nicht erreichbar (z. B. DS-Lite / CGNAT).
> - PublicKey oder PresharedKey stimmen zwischen Server und Client nicht exakt überein.
> - Bei Clients hinter NAT kann ein fehlendes `PersistentKeepalive = 25` dazu führen, dass eine zuvor funktionierende Verbindung nach längeren Idle-Phasen einschläft und von außen nicht mehr unmittelbar erreichbar ist.

2. Routing-Prüfung

Besteht ein aktueller Handshake, teste das Routing:

```
# Gateway anpingen
ping -c 3 10.8.0.1

# Externe IP anpingen (Prüft Forwarding und Masquerading)
ping -c 3 1.1.1.1

# Routing-Tabelle für eine Ziel-IP abfragen
ip route get 1.1.1.1
```

Wird `10.8.0.1` erreicht, aber `1.1.1.1` nicht, ist das Kernel-Forwarding (`sysctl net.ipv4.ip_forward`) oder die UFW/NAT-Regel auf dem Server fehlerhaft.

3. MTU & Path-MTU-Discovery verstehen

`wg-quick` ermittelt die MTU des WireGuard-Tunnels standardmäßig automatisch anhand des Routings zum Endpoint beziehungsweise der MTU der zugrunde liegenden Schnittstelle.

Eine Tunnel-MTU von 1420 Bytes ist bei klassischen Setups mit einem standardmäßigen 1500-Byte-Ethernet-Underlay häufig anzutreffen, stellt jedoch keinen universellen WireGuard-Wert dar:

Abweichende Underlays:

Anschlüsse mit PPPoE (typischerweise MTU 1492), zusätzliche Kapselungen (DS-Lite, GRE, VXLAN), Mobilfunk (LTE/5G) oder verschachtelte VPNs verringern die tatsächliche Path MTU (PMTU) der Verbindung.

Manuelle MTU-Konfiguration:

Eine manuelle `MTU =`-Angabe in der Client- oder Serverkonfiguration sollte nur erfolgen, wenn die automatische Erkennung fehlschlägt oder konkrete PMTU-Probleme diagnostiziert wurden (z. B. Hängenbleiben bei großen Webseiten oder SSH-Transfers).

PMTU-Diagnose mit Ping:

Mit dem Ping-Befehl und gesetztem Don't-Fragment-Flag (`-M do`) lässt sich testen, bis zu welcher Größe unfragmentierte IP-Pakete über die Strecke transportiert werden können. Beachte dabei, was gemessen wird: Bei IPv4 setzt sich die gesamte IP-Paketgröße aus der ICMP-Payload und 28 Bytes Headern (20 Bytes IPv4-Header + 8 Bytes ICMP-Header) zusammen:

$$\text{IP-Paketgröße} = \text{Payload} + 28\text{ Bytes}$$

```
# Testet ein IP-Paket von 1420 Bytes (1392 Bytes Payload + 28 Bytes Header)
ping -M do -s 1392 -c 3 <SERVER_IP>
```

Wird dieser Test mit `Frag needed and DF set` abgewiesen, verringere die Payload in 8er-Schritten (z. B. auf 1364 oder 1332), um die maximale unfragmentierte Paketgröße auf diesem Pfad zu

ermitteln.

> **Wichtiger Hinweis zur MTU-Ermittlung:** Das Messergebnis spiegelt die MTU des Transportpfades wider und lässt sich nicht 1:1 als universelle WireGuard-MTU übernehmen. Es liefert bei Übertragungsproblemen jedoch den entscheidenden Anhaltspunkt zur gezielten Anpassung des `MTU =`-Wertes im Client-Profil unter `[Interface]`.

DNS-Leak-Test & Verschlüsselung

Wenn der VPN-Tunnel aktiv ist, müssen alle DNS-Anfragen über das VPN geleitet werden. Anderenfalls fließen DNS-Anfragen am Tunnel vorbei unverschlüsselt an den lokalen Internet Service Provider (DNS-Leak).

DNS-Leak testen

Schritt 1: Tunnel-Verbindung herstellen

Aktiviere den WireGuard-Tunnel auf dem Client:

```
sudo wg-quick up wg0
```

Schritt 2: Lokalen Systemresolver prüfen

Verifiziere im Terminal, welche DNS-Server und Routing-Domains der Systemresolver tatsächlich für das Tunnel-Interface nutzt:

```
# Status und zugewiesene Resolver des Tunnel-Interfaces prüfen
resolvectl status wg0

# Testabfrage über den konfigurierten Resolver durchführen
resolvectl query example.com
```

Schritt 3: Externe Leak-Erkennung durchführen

Teste extern, über welche Resolver-Infrastruktur deine Anfragen im Web ankommen (ohne manuell einen Resolver per `@` zu erzwingen):

```
dig +short TXT whoami.ds.akahelp.net
```

Öffne alternativ im Browser Testseiten wie `https://dnsleaktest.com` oder `https://ipleak.net`.

Schritt 4: Testergebnis bewerten

Ein DNS-Leak wird nicht allein durch die Eigentümerschaft eines Resolvers definiert, sondern dadurch, dass DNS-Anfragen einen unerwarteten Resolverpfad außerhalb der vorgesehenen VPN-/DNS-Architektur verwenden. Bei der hier beschriebenen Konfiguration sollten DNS-Anfragen nicht über den Resolver des lokalen Internet-Providers (ISP) laufen. Wird dieser dennoch verwendet, obwohl Unbound beziehungsweise der konfigurierte VPN-Resolver vorgesehen ist, deutet dies auf einen DNS-Leak oder eine fehlerhafte Resolver-Priorisierung auf dem Client hin.

Werden öffentliche Anycast-Resolver (wie Quad9 oder Cloudflare) genutzt, müssen diese geografisch nicht am selben Standort wie dein VPN-Gateway antworten, dürfen jedoch keinesfalls unverschlüsselt über die physische Verbindung des lokalen Internetanschlusses angefragt werden.

> ⚠ **Tipp bei DNS-Leaks:** Falls lokale DNS-Server des Heimnetz-Routers angezeigt werden, prüfe, ob `systemd-resolved` oder der lokale Verbindungsmanager die Priorisierung des WireGuard-Interfaces korrekt vorgenommen hat.

nftables-Alternative für Fortgeschrittene

Für Administratoren, die anstelle von UFW ein rein natives `nftables`-Regelwerk pflegen möchten, bietet nftables atomare Updates und eine gemeinsame Syntax für IPv4 und IPv6.

Vollständige Host-Firewall mit WireGuard (`/etc/nftables.d/wireguard.nft`)

Das folgende Regelwerk fungiert als vollständige, restriktive Host-Firewall (`policy drop`). Es erlaubt Loopback, bestehende Verbindungen, Netzwerkdiagnose via ICMP/ICMPv6, SSH-Administration sowie den WireGuard-Tunnelverkehr. Sofern die Administrationsarchitektur dies zulässt, sollte SSH zusätzlich auf bekannte Management-Netze beziehungsweise vertrauenswürdige Quelladressen beschränkt werden:

```
sudo mkdir -p /etc/nftables.d

sudo tee /etc/nftables.d/wireguard.nft << 'EOF'
#!/usr/sbin/nft -f

# Vollständige Host-Firewall inklusive WireGuard-VPN und NAT
table inet filter {
    chain input {
        type filter hook input priority filter; policy drop;

        # Lokales Loopback-Interface erlauben
        iif "lo" accept comment "Loopback erlauben"

        # Bestehende und zugehörige Verbindungen erlauben
        ct state established,related accept comment "Established/Related erlauben"
```

```
        # Ungültige Pakete verwerfen
        ct state invalid drop comment "Ungültige Pakete verwerfen"

        # ICMP und ICMPv6 für Netzwerkdiagnose und Path-MTU-Discovery erlauben
        ip protocol icmp accept comment "IPv4 ICMP (Ping, PMTU) erlauben"
        ip6 nexthdr icmpv6 accept comment "IPv6 ICMPv6 (NDP, Ping, PMTU) erlauben"

        # SSH-Zugang für Administration (Port anpassen falls abweichend)
        tcp dport 22 accept comment "SSH Management erlauben"

        # Eingehenden WireGuard VPN-Port erlauben
        udp dport 51820 accept comment "WireGuard VPN Port erlauben"

        # Lokalen DNS-Zugriff (Unbound) für WireGuard-Clients erlauben (UDP & TCP)
        iifname "wg0" udp dport 53 accept comment "DNS from WireGuard"
        iifname "wg0" tcp dport 53 accept comment "DNS from WireGuard"
    }

    chain forward {
        type filter hook forward priority filter; policy drop;

        # Bestehende Verbindungen im Forwarding erlauben
        ct state established,related accept comment "Forward Established/Related"

        # Datenverkehr aus dem VPN ins Internet (eth0) weiterleiten
        iifname "wg0" oifname "eth0" accept comment "WireGuard Client Forwarding"
    }
}

table inet nat {
    chain postrouting {
        type nat hook postrouting priority srcnat; policy accept;

        # IPv4 Masquerading (NAT) für WireGuard-Subnetz über WAN
        ip saddr 10.8.0.0/24 oifname "eth0" masquerade comment "WireGuard IPv4 NAT"

        # Optionales IPv6 NAT66 (nur bei ULA-Konfiguration notwendig)
        # ip6 saddr fd42:42:42::/64 oifname "eth0" masquerade comment "WireGuard IPv6
NAT66"
    }
}
EOF
```

```
sudo chmod 600 /etc/nftables.d/wireguard.nft
```

Einbindung in `/etc/nftables.conf`

Damit das Regelwerk beim Systemstart geladen wird, binde das Verzeichnis in `/etc/nftables.conf` ein:

```
if ! grep -q 'include "/etc/nftables.d/\*.nft"' /etc/nftables.conf; then
    echo 'include "/etc/nftables.d/*.nft"' | sudo tee -a /etc/nftables.conf
fi

# Konfiguration prüfen und laden
sudo nft -f /etc/nftables.conf
sudo systemctl enable --now nftables
```

Verifikation:

```
sudo nft list ruleset
```

Firewall-Sets für granulare Zugriffskontrollen (ACLs)

Sets in nftables sind nützlich, wenn du nicht pauschal allen Peers dieselben Rechte geben möchtest, sondern Zugriffskontrolllisten (ACLs) pflegst (z. B. dürfen nur bestimmte Admin-Peers auf interne SSH-Dienste zugreifen):

```
# Definiere ein Set für privilegierte Peers in der Tabelle 'filter'
sudo nft add set inet filter admin_peers '{ type ipv4_addr; }'
sudo nft add element inet filter admin_peers '{ 10.8.0.2, 10.8.0.5 }'

# Erlaube nur Mitgliedern des Sets Zugriff auf SSH im internen Management-Netz
sudo nft add rule inet filter forward ip saddr @admin_peers ip daddr 192.168.10.10 tcp
dport 22 accept
```

Logging & Fehlerdiagnose

1. Dynamisches Kernel-Debugging temporär aktivieren

WireGuard arbeitet im Regelbetrieb geräuschlos und erzeugt standardmäßig keine Meldungen im Kernel-Log. Bei schwierigen Verbindungsabbrüchen kannst du das Kernel-Debugging aktivieren:

```
# Debug-Logging für das Kernelmodul einschalten
echo "module wireguard +p" | sudo tee /sys/kernel/debug/dynamic_debug/control

# Meldungen live im Systemprotokoll verfolgen
sudo dmesg -wT | grep -i wireguard
```

Nach Abschluss der Fehleranalyse solltest du das Logging wieder deaktivieren, um Protokolloverhead zu vermeiden:

```
echo "module wireguard -p" | sudo tee /sys/kernel/debug/dynamic_debug/control
```

2. Systemd-Protokolle und Schnittstellen analysieren

```
# Systemd-Ereignisse von wg-quick auslesen
journalctl -u wg-quick@wg0 --since "1 hour ago"

# Maschinenlesbare Zusammenfassung aller Peers
sudo wg show wg0 dump

# Socket-Status und Warteschlangen prüfen
sudo ss -ulnp 'sport = :51820'
```

Befehlsreferenz (Cheatsheet)

Befehl	Kategorie	Zweck und betriebliche Wirkung
`sudo systemctl start wg-quick@wg0`	Dienststeuerung	Startet das Interface `wg0` und richtet Tunnel-Routen ein
`sudo systemctl stop wg-quick@wg0`	Dienststeuerung	Beendet das Interface und baut Tunnel-Routen ab
`sudo systemctl enable --now wg-quick@wg0`	Autostart	Aktiviert den automatischen Systemstart des VPN-Tunnels
`sudo wg show`	Monitoring	Zeigt aktive Peers, Handshake-Zeiten, Endpunkte und Transfervolumen

Befehl	Kategorie	Zweck und betriebliche Wirkung
`sudo wg show wg0 dump`	Scripting / API	Gibt tabellarische, maschinenlesbare Peer- und Tunneldaten aus
`sudo wg syncconf wg0 <(wg-quick strip wg0)`	Live-Reload	Gleicht Peer-Änderungen im laufenden Kernel ab, ohne aktive Tunnels zu trennen
`wg genkey	tee priv.key	wg pubkey > pub.key`	Kryptografie	Erzeugt ein asymmetrisches Curve25519-Schlüsselpaar
`wg genpsk > preshared.key`	Kryptografie	Erzeugt einen symmetrischen 256-Bit Pre-Shared Key (Zusatzschutz)
`qrencode -t ansiutf8 < client.conf`	Deployment	Rendert eine Konfigurationsdatei als scanbaren Terminal-QR-Code
`sudo add-wireguard-client <name> [suffix]`	Automatisierung	Erzeugt vollautomatisch ein neues Profil fehlertolerant mit File-Locking und Rollback
`sudo ufw route allow in on wg0 out on eth0`	Firewall	Erlaubt gezieltes Forwarding von WireGuard-Clients über die WAN-Schnittstelle
`sudo ufw status verbose`	Firewall	Zeigt UFW-Status und Portfreigaben detailliert an
`sudo nft -f /etc/nftables.conf`	Firewall	Lädt das native nftables-Regelwerk atomar
`echo "module wireguard +p"	sudo tee ...`	Troubleshooting	Aktiviert dynamisches Kernel-Debugging für detaillierte Diagnose
`resolvectl status wg0`	DNS	Zeigt DNS-Resolver-Zuweisung und DNSSEC auf der Schnittstelle an

Weiterführende Ressourcen

Ressource	Link	Zweck und Inhalt
WireGuard Offiziell	WireGuard Technical Whitepaper	Offizielles Protokolldesign, Noise-Framework und Krypto-Analyse

Ressource	Link	Zweck und Inhalt
WireGuard Tools & Quick	WireGuard Manpages	Offizielle Referenz zu Parametern von `wg` und `wg-quick`
Ubuntu Server Guide	Ubuntu Server Documentation	Leitfaden für Netzwerkkonfiguration, Kernel-Routing und UFW
Linux Server Härtung	Linux Server Härtung: SSH & CrowdSec	Sicherheitsrichtlinien, SSH mit FIDO2 und Firewall-Härtung
Ubuntu Upgrade Leitfaden	Ubuntu 24.04 auf 26.04 LTS Upgrade	System-Upgrade von Kernel und Paketquellen auf die nächste LTS-Version
LPIC-1 Kursserie	LPIC-1 Serie: Linux-Administration	Fundierte Kursgrundlagen zu Subnetzen, Routing und Berechtigungen

Fazit

WireGuard bietet eine moderne, performante VPN-Architektur für Linux-Server: schlank im Code, direkt als Kernel-Modul implementiert und mit kryptografischer Präzision auf Basis moderner Primitives (Curve25519, ChaCha20-Poly1305, BLAKE2s). Die direkte Unterstützung in aktuellen LTS-Releases von Ubuntu gewährleistet eine stabile und wartungsarme Basis für Gateways und Standortkopplungen.

Die wichtigsten betrieblichen Erkenntnisse im Überblick:

Protokolldesign & Handshake:

WireGuard verzichtet auf schwerfällige Userspace-Verbindungsdaemons. Der 1-RTT-Handshake basiert auf dem Noise-Protokollframework und wird direkt im Kernel abgewickelt, was Verbindungsaufbauten beschleunigt und nahtloses Roaming ermöglicht.

Kryptografische Einordnung: Symmetrische Pre-Shared Keys (`PresharedKey`) bieten wertvollen Schutz vor rückwirkender Entschlüsselung („Harvest Now, Decrypt Later“), machen das Setup jedoch nicht zu einem vollständigen post-quanten-sicheren KEM-Verfahren.

Architektur bei IPv4 und IPv6: IPv4 arbeitet standardmäßig mit privatem Subnetz und NAT. Für IPv6 ist ein geroutetes GUA-Präfix der bevorzugte Standard; ULA mit NAT66 dient als

zweckmäßige Alternative, wenn Provider kein Client-Präfix routen.

Konsistente Firewall-Steuerung: Vermeide unkontrollierte globale Freigaben (`DEFAULT_FORWARD_POLICY="ACCEPT"`). Pflege Forwarding und NAT strukturiert in den UFW-Regeldateien oder über ein eigenständiges `nftables` -Regelwerk.

Unterbrechungsfreier Betrieb: Die Aktualisierung von Peers über `wg syncconf` schützt aktive VPN-Sessions vor Verbindungsabbrüchen und ermöglicht saubere Automatisierungen im Produktivbetrieb.

Beginne bei neuen Setups mit einem einzelnen Client und verifiziere Handshake (`wg show`), IP-Routing (`ping`) und DNS-Auflösung (`resolvectl status`) systematisch als getrennte Ebenen, bevor weitere Peers ausgerollt werden.