Wireguard

> Ein **VPN** mit Wireguard werden oftmals unter Zuhilfenahme von **wg-quick** aufgebaut, die Verwendung eigene Scripts kann dennoch sinnvoller sein.

1) VPN Betriebs-Modus

Ein VPN kann sowohl als VPN wie als Full-VPN konfiguriert werden.

- VPN bedeutet, dass nur der Netzverkehr zur Gegenstelle oder im Netz der Gegenstelle über den Tunnel läuft.
- Full-VPN bedeutet, dass den gesamten Netzwerk-Verkehr durch den Tunnel geleitet wird.

1.1) Problematik

- Die Betriebsmodus (VPN oder Full-VPN) bedürfen getrennte Konfigurationsdateien.
- Full-VPN geht nicht ganz wie erwünscht.
    - Falls die IPv4-Adressen des Heim-LANs mit de des Gast-Netzes kollidieren und die Verbindung zum VPN über IPv4 geschieht, kann es zu Störungen kommen.
    - Die von **wg-quick** eingefügte Routen sorgen bestimmt, dass die Daten vom Tunnel nicht zum eigentliche Empfänger geleitet werden.

2) Routing unter Linux

> Neuere Linux Distributionen verwenden das NetworkManager um IP-Adressen und Route festzulegen (auf ein Raspberry wird zurzeit dhcpcd verwendet).

Ein saubere Routing setz eine Bevorzugung der eine oder andere Netzwerk-Schnittstelle voraus.

Das Zauberwort heißt **metric**:

```
1 $ ip route show
2 ...
3 default via 10.0.0.1 dev eth0 proto dhcp metric 600
4 10.0.0.0/24 dev eth0  ... metric 100
5 10.0.0.0/24 dev wlan0 ... metric 600
6 10.0.0.0/24 dev wg0   ...
7 ...
```

Der Eintrag *metric 100* bei der Schnittstelle eth0 hat ein geringerer Wert als die der Schnittstelle wlan0 und wird bevorzugt. Eine Funkverbindung fügt Verzögerungen, diese bewirken sich negativ auf der Übertragungsrate, deswegen wird eine höhere *metric* der Route über das Gerät wlan0, hier, vom NetworkManager zugewiesen.

In der 5. Zeile sehen wir ein Eintrag ohne metric, es ist gleichbedeutend mit **metric 0**. Mit das Kommando **route** ist es ersichtlich

```
$ route
Kernel IP Routentabelle
Ziel          Router        Genmask         Flags Metric Ref  Use Iface
...
default       10.0.0.1      255.255.255.255 U     100    0      0 eth0
10.0.0.0      0.0.0.0       255.255.255.0   U     100    0      0 eth0
10.0.0.0      0.0.0.0       255.255.255.0   U     600    0      0 wlan0
10.0.0.0      0.0.0.0       255.255.255.0   U     0      0      0 wg0
```

Netzwerkzugriffe auf beispielsweise 10.0.0.2 würden, hier der Weg über wg0 nehmen.

Schlimmer ist, dass der Zugriff zur Adresse unseren VPN-Endpunkt nicht der Weg über eth0 nehmen kann! (FUll-VPN Betrieb).

2.1) Routing und Spezialität

```
$ ip route show
default via 10.0.0.1 dev eth0 proto dhcp
10.1.1.0/24 via 10.1.1.1 dev wlan0
10.0.0.1/32 via 10.1.1.1 dev wlan0
```

In diesem Fall ist die Route zu 10.0.0.1 über die Netzwerk-Schnittstelle wlan0 genauer spezifiziert und wird als ersten beachtet.

Netzwerk Pakete die an Adressen im Bereich 10.1.1.0/24 gerichtet werden nehmen den Weg über wlan0. 10.1.1.0/24 ist spezifischer als 0.0.0.0/0 (default).

2.2) Verhalten bei IPv4 und IPv6 im Full-VPN Modus - Konfiguration

```
[Interface] # client
PrivateKey =  W...=
Address = 10.18.1.2/32, fd01:cafe::2/128
DNS = 192.168.178.1 fritz.box 178.168.192.in-addr.arpa
MTU=1420
Table = auto

[Peer] # Server
PublicKey  =  7...=
AllowedIPs = 198.18.1.0/24, 192.168.178.0/24, fd01:cafe::/64, 0.0.0.0/0, ::/0
EndPoint   = 192.0.2.12
```

Zeile 3 definiert die Adressen der Wireguard Netzwerkschnittstelle.

Zeile 4 sorgt dafür, dass der DNS-Server (in dem Fall eine Fritz!Box) im Heim-LAN immer konsultiert wird.

Die Zeile **Table = auto** ist hier überflüssig, wenn es auf **off** gestellt wird, wird keine zusätzliche Routingtabelle erstellt

Zeile 10 sorgt dafür, dass auf der Client Routen entsprechend gesetzt werden.

0.0.0.0/0 und ::/0 stellen sicher (oder sollten es bei IPv6), dass der Verkehr zur Wireguard Netzwerkschnittstelle geleitet werden.

2.3) Route auf der Client

```
$ ip route
default via 10.0.0.1 dev clan0 proto dhcp metric 600
10.0.0.0/24 dev tun0 scope link
10.0.0.0/24 dev wlan0 proto kernel scope link src 10.0.0.3 metric 600
10.18.1.0/24 dev tun0 scope link
```

So ist nicht ersichtlich dass die default Route durch eine andere ergänzt wurde.

```
ip -4 route show table all
default dev tun0 table 51820 scope link
default via 10.0.0.1 dev wlan0 proto dhcp metric 600
10.0.0.0/24 dev tun0 scope link
10.0.0.0/24 dev tun0 proto kernel scope link src 10.0.0.3 metric 600
10.18.1.0/24 dev test scope link
...
```

Hier offenbart was geschieht. Eine Routing Tabelle mit ID 51820 wurde angelegt, dort steht eine default Route, die höherprior ist als die in Zeile 3.

3) Client

> Wer lesen kann ist im Vorteil.

In dem Fall werden Manual Seiten und ein wenig Scripts.

3.1) Wireguard Tunnel erzeugen

```
ip tunnel add dev tun0 type wireguard
```

wg das Verwaltung Programm (Kommunikation zur Wireguard) kann erst dann aufgerufen werden, wenn eine passenden Schnittstelle vorhanden ist.

3.2) wg starten

```
wg up tun0 wg-client.conf
```

Über der Konfigurationsdatei wg-client.conf wird der Kernel parametriert.

3.3) Beispiel einer Konfigurationsdatei

```
[Interface] # mobile Gerät
PrivateKey =  W...=

[Peer] # Server
PublicKey  =  7...=
AllowedIPs = 0.0.0.0/0, ::/0
EndPoint   = 192.0.2.21:51820
PersistentKeepAlive = 25
```

Der Rest kann über Script konfiguriert werden. Die Zuweisung *EndPoint = 192.0.2.21:51820* könnte auch in unseren Script statt finden.

3.4) Tunnel einschalten, IP Adresse und Route Setzen

```
1 ip link set mtu 1480 up dev tun0
2 ip address add 10.0.0.2/24 dev tun0
```

In Zeile 1 wird Die Schnittstelle parametriert und hochgefahren.
mtu 1480 gibt an wie groß die einzelnen Netzwerkpakete sein dürfen. Wireguard benötigt selbst Platz in jedes Paket, dies vermindert die zulässige “mtu”.

Die 2. Zeile liegt die IPv4 Adresse der Schnittstelle und auch eine Route.
/24 besagt, dass Pakete zum Ziel 10.0.0.0 bis 10.0.0.255 über tun0 geleitet werden.

3.5) Route zum Heim LAN setzen

```
1 ip route add 192.168.178.0/24 via 10.0.0.2 dev tun0
```

Rechner in unseren Heim-LAN haben alle eine Adresse im Bereich 192.168.178.0/24. Wir können hiermit jedes Gerät gezielt ansprechen.

Wenn die Gastgeberin ebenfalls eine Fritz!Box betreibt, gibt es Kollisionen. Später mehr dazu.

3.6) DNS konfigurieren

```
1 echo nameserver 192.168.178.1 | resolvconf -a tun0 -m 0 -x
2 DEV=$(ip route get 1.1.1.1 | awk '{print $5;exit}')
3 resolvectl default-route $DEV false
4 resolvectl domain tun0 friz.box 178.168.192.in-addr.arpa
```

DNS-Anfrage werden vorzugsweise über tun0 gesendet, damit können wir die Systeme im Heim-LAN per Name ansprechen. die entsprechenden DNS Anfragen werden von unseren eigenen Nameserver bedient.

Da wir über WIFI oder per Kabel mit das Netz der Gastgeberin verbunden sein können, wird in Zeile 2 über welche Schnittstelle eine öffentliche Adresse erreicht wird. Danach werden die Anfragen für die Domäne Fritz!Box und für unseren heimischen IP-Bereich den Weg über unseren Tunnel verwiesen (Zeile 4).

Die Anweisungen in Zeile 2 und 3 können schlecht mit wg-quick implementiert werden.

3.7) Client Script Konfigurationsdatei

```
WG=tun0
CNF=/etc/wireguard/tun0.conf
N=2
IP4=10.18.1.$N
IP6=fd01:cafe::$N
DNS=192.168.178.1
DOMAIN="fritz.box 178.168.192.in-addr-arpa"
MTU=1420
IPR=192.168.178.0/24
EP4=192.0.2.22
EP6=2001:db8::123
URL=
PORT=51820
```

Falls mehrere Systeme im VPN eingebunden werden, reich es aus die Variable N zu setzen. Die übrige Variable können wir später im Startscript verwenden.

3.8) Client Script

```
#!/bin/sh
source /etc/wireguard/tun0.var
getDevViaDrt() {
    VIA=$(ip route get 1.1.1.1 | awk '{print $3;exit}')
    DEV=$(ip route get 1.1.1.1 | awk '{print $5;exit}')
    DRT=$(ip route | grep 'default' | awk '{print $1;exit}')
}
case $1 in
up)
    getDevViaDrt
    ip link add dev $WG type wireguard
    wg setconf $WG $CNF
    ip link set mtu $MTU up dev $WG
    ip address add $IP4/24 dev $WG
    ip route add $IPRN via $IP4 dev $WG
    ip route add $EP4 via $VIA dev $DEV
    echo nameserver $DNS | resolvconf -a $WG -m 0 -x
    resolvectl  default-route $DEV false
    resolvectl domain $WG $DOMAIN
    resolvectl default-route $WG
    ;;
down)
    ip link del dev $WG
    getDevViaDrt
    ip route del $EP4 via $VIA dev $DEV
    resolvectl default-route $DEV true
    ;;
esac
```

In Zeile 3 sorgt Funktion getDevVia(), dass wir erfahren wie die Pakete ins Netzwerk gesendet werden (Aufruf Zeile 9).

EP4 ist die IPv4 Adresse vom Tunnel Endpunkt. Ws muss sichergestellt werden, dass der Netzwerkverkehr zum Endpunkt nicht fehlgeleitet wird. Dies erfolgt in Zeile 16.

Anschließend wir das DNS konfiguriert

Zeile 24 entfernt die VPN Schnittstelle und auch vorhergehende Anweisungen, die sich auf der Schnittstelle bezogen hatten. Eine Überprüfung, ob es notwendig ist, wurde nicht vorgenommen, diese Route bedeutet kein Nachteil.

Im Block “down” wird das VPN abgeschaltet und aufgeräumt.

3.9) Endpunkt über IPv4 oder IPv6 - Full VPN steuern

```
1  #!/bin/bash
2  error() {
3     echo Syntax: $(basename $0) "[-f] [-6] up|down"
4  }
5  V6=false
6  FULL=false
7  CMD=
8  while [[ $# -gt 0 ]]
9  do
10     case $1 in
11     -6) V6=true;;
12     -f) FULL=true;;
13     up)  CMD=$1;;
14     down) CMD=$1;;
15     *) error; exit 1;
16     esac
17     shift 1
18 done
```

Mit dieser Code könnten Aufruf Optionen zur Steuerung unseren Script ausgewertet werden. Es wurde hier nicht implementiert.

3.10) Ergänzungen für IPv6 Endpunkt

```
1  source /etc/wireguard/tun0.var
2  KEY=$(sed -n -e '/PublicKey/p' $CNF | tr -d ' ' | sed 's/PublicKey=//')
3  ...
4      wg setconf $WG $CNF
5      case $V6 in
6      true) wg set $WG peer "$KEY" endpoint "[$EP6]:$PORT";;
7      esac
8  ...
```

Die Variable PORT haben wir in der Datei /etc/wireguard/tun0.var definiert, ebenso EP4 und EP6 (Tunnel Endpunkt).

3.11) IPv6 des Endpunktes aus eine URL entnehmen

```
1  EP6=$(host $URL | grep ':' | awk '{print $NF}')
```

Alternativ

```
EP6=$(nslookup $URL | grep Address: |\
       grep -v '#' | awk ' { print $NF}'| grep ':')
```

Alternativ

```
EP6=$(dig -q $URL  -t AAAA | egrep -v ';|^$' |\
       awk '{print $NF}')
```

Wenn ein DynDNS angewendet wird kann die Adresse des Endpunktes ermittelt werden, dies kann von Vorteil sein, ein VPS ist nicht unbedingt notwendig.

Die URL, beispielsweise wireguard.example.org, muss in unsere Script Konfigurationsdatei zugewiesen sein.

3.12) Full VPN

> Ein Full-VPN Verbindung bedeutet, dass alle Kommunikationen mit der weiten Welt ausschließlich über der VPN-Verbindung laufen.

3.13) Sicherstellen, dass das VPN Tunnel der richtige Weg nimmt

Im Full-VPN Betrieb muss eine weitere Route eingestellt werden, entsprechend der gesetzte Routen (unseren Script und auch wg-quick) werden alle Paketen über der Wireguard Schnittstelle geleitet.

Wir müssen dementsprechend die Route zum Endpunkt explizit setzen, damit der Verkehr zum Endpunkt den Weg über der eigentlichen Schnittstelle nimmt.

```
1  source /etc/wireguard/tun0.var
2
3  AEP4=$(wg | sed -n 's/.*endpoint: \([^[].*\):.*/\1/p')
4  AEP6=$(wg | sed -n 's/.*endpoint: \[\(.*:.*:.*\)].*/\1/p')
5
6  GW=$(ip route get 1.1.1.1 | awk '{print $3;exit}')
7  DEV=$(ip route get 1.1.1.1 | awk '{print $5;exit}')
8  LL=$(ip -6 route | grep default | awk '{print $3}')
9
10 ip add address $IP6/64 dev $WG metric 0
11 ip add address default dev $WG metric 0
12
13 if [[ "$AEP4" != "" ]]; then
14     ip route add $AEP4/32 via $GW dev $DEV
15 fi
16 if [[ "$AEP6" != "" ]]; then
17     ip -6 route add $AEP6/128 via $LL dev $DEV
18 fi
```

Im Code, es könnte ein getrennten Script sein, wird die aktuellen Tunnel Endpunkt Adresse ermittelt. Hier wird von einer Konfigurierung mit IP-Adressen ausgegangen.

In der Zeile 4 bzw, 5 wird die Adresse des aktuellen Endpunktes ermittelt. Eine der beiden Variable wird mit einer IP-Adresse gesetzt, die andere Variable ist leer.

Sollte dennoch eine URL als Endpunkt verwendet worden sein, kann die Adresse wie unter **IPv6 des Endpunktes aus eine URL entnehmen** beschrieben ermittelt werden.

Da wir noch nicht das Full-VPN Betrieb eingestellt haben, können die notwendigen Informationen zum Setzen der neuen Routen in Zeilen 7 bis 8 geholt werden.

Anschließend wird eine spezifische Route zum Endpunkt gesetzt, damit ist sichergestellt, dass unsere VPN-Verbindung immer den richtigen Weg nimmt.

4) Erweiterten Beispiel

Namenskonvention

- GUI: **INTERFACE_NAME**-gui.sh
- SCRIPT: **INTERFACE_NAME**.sh
- CONFIGURATION: **INTERFACE_NAME**.conf
- VARIABLEN: **INTERFACE_NAME**.var
- KONFIG: **INTERFACE_NAME**.conf
- DEV: **INTERFACE_NAME**

Die Namensgebung der Scripten geben sind auf der Name der Tunnel-Schnittstelle abgestimmt. Damit können wir verschiedene VPN-Tunnel einfach verwalten.

4.1) Erweiterten Beispiel

Beispiel für Dateien

Falls wir 2 mögliche Tunnel verwenden wollen (nicht gleichzeitig), den einen zu Anna und den anderen zu Bernd, hätten wir nachstehenden Dateien.

```
/etc/wireguard/anna.conf
/etc/wireguard/anna.var
/usr/local/bin/anna.sh
/usr/local/bin/anna-gui.sh

/etc/wireguard/bernd.conf
/etc/wireguard/bernd.var
/usr/local/bin/bernd.sh
/usr/local/bin/bernd-gui.sh
```

4.2) /etc/wireguard/anna.conf

```
1 [Interface] # client
2 PrivateKey =  mOLMqQ3XujHfR+I7l5Cbem6kRB77njYBoVz8l2mR5Xk=
3
4 [Peer] # Server
5 PublicKey  =  wcnpq2hRI1Pzd1VfSLndg1P4v8qBVAb8P+W0MrkX0CY=
6 AllowedIPs = 10.18.1.32/32, 0.0.0.0/0, ::/0
7 PersistentKeepAlive = 25
```

4.3) /etc/wireguard/anna.var

```
N=2
IP4=10.18.1.$N
IP6=fd01:cafe::$N
DNS=192.168.178.1
DOMAIN="fritz.box 178.168.192.in-addr-arpa"
MTU=1420
IPR=192.168.178.0/24
EP4=192.0.2.223
EP6=2001:db8:dead:beef::1
URL=
PORT=51820
```

4.4) /etc/wireguard/bernd.var

```
N=6
IP4=172.17.1.$N
IP6=fdab:affe::$N
DNS=172.17.1.1
DOMAIN="dslrouter 1.168.192.in-addr-arpa"
MTU=1420
IPR=192.168.1.0/24
EP4=
EP6=
URL=wireguard.example.org
PORT=51820
```

4.5) /usr/local/bin/wg-anna.sh

Das Script sorgt dafür, dass das gewünschten Tunnel aufgebaut wird, der Name der Konfigurationsdateien und der Schnittstellen-Name werden aus der Name des Scripts entnommen.

Die einzelnen Phasen des Aktivierens oder Stoppens sind in kleine Funktionen unterteilt, damit ist der Hauptcode lesbarer.

```
#!/bin/sh
error() {
   echo Syntax: $(basename $0) "[-f] [-6] [-c conf] up|down"
}
V6=false
FULL=false
CMD=
CONF=/etc/wireguard/$(basename $0 .sh)

while [[ $# -gt 0 ]]
do
    case $1 in
    -6) V6=true;;
    -f) FULL=true;;
    -c) CONF=$2; shift 1;;
    up)  CMD=$1;;
    down) CMD=$1;;
    *) echo Wrong parameter $1;
       error Parameter $1; exit 1;
    esac
    shift 1
done

source ${CONF}.var
CNF=${CONF}.conf
WG=$(basename $CONF)

addFullVPN() {
   if [[ "$FULL" == true ]]; then
        ip add add $IP6/0 dev $WG metric 10
        ip route add $EP6 via $LL dev $DEV
        ip route add default via $IP4 dev $WG
    fi
}

delFullVPN() {
    if [[ "$EP6" != "" ]]; then
        ip route del $EP6 via $LL dev $DEV
    fi
}

getDevViaScr() {
    VIA=$(ip route get 1.1.1.1 | awk '{print $3;exit}')
    DEV=$(ip route get 1.1.1.1 | awk '{print $5;exit}')
    SRC=`ip route get 1.1.1.1 | awk '{print $7;exit}'`
    LL=$(ip -6 route | grep default | awk '{print $3}')
}

getEP() {
    if [[ "$URL" != "" ]]; then
        EP4=$(host $URL | grep -v ':' | awk '{print $NF}')
        EP6=$(host $URL | grep ':' | awk '{print $NF}')
    fi
}
```

```
55
56  addDNS() {
57      echo nameserver $DNS | resolvconf -a $WG -m 0 -x
58      resolvectl  default-route $DEV false
59      resolvectl domain $WG $DOMAIN
60      resolvectl default-route $WG
61  }
62
63  case $CMD in
64  up)
65      getDevViaScr
66      getEP
67      ip link add dev $WG type wireguard
68      wg setconf $WG $CNF
69      ip link set mtu $MTU up dev $WG
70      KEY=$(sed -n -e '/PublicKey/p' $CNF | tr -d ' ' | sed 's/PublicKey=//')
71      if [[ "$V6" == true ]]; then
72          wg set $WG peer "$KEY" endpoint "[$EP6]:$PORT"
73      else
74          wg set $WG peer "$KEY" endpoint "$EP4:$PORT"
75      fi
76      ip address add $IP4/24 dev $WG
77      ip route add $IPR via $IP4 dev $WG
78      ip route add $EP4 via $VIA dev $DEV metric 3
79      addDNS
80      addFullVPN;
81      ;;
82  down)
83      ip link del dev $WG
84      getDevViaScr
85      EP4=$(ip route show | grep 'metric 3' | awk '{print $1}')
86      if [[ "$EP4" ]]; then
87           ip route del $EP4 via $VIA dev $DEV
88      fi
89      resolvectl default-route $DEV true
90      delFullVPN
91      ;;
92  esac
```

4.6) GUI

```
1  #!/bin/bash
2  SCRIPTPATH=~/bin
3  #C=-c
4  #P="~/wireguard/wg-client"
5  TUN=$(basename $0 -gui.sh
6  if ip add show $DEV
7    TEXT=Stop; CMD="down"
8  else
9    TEXT=Start; CMD="up"
10   ENTRY1='--field=''Ipv6 Endpunkt'':CHK'; ENTRY2='--field=''Full VPN'':CHK'
11 fi
12 IN=$(yad --title Wireguard --no-escape \
13     --text="$TEXT $TUN" --image="dialog-password" \
14     --form  --field="Passwort":H "$ENTRY1" "$ENTRY2")
15 if [[ $? -eq 0 ]]; then
16    PWD=$(echo $IN | awk -F '|' '{print $1}')
17    X=$(echo $IN | awk -F '|' '{print $2}')
18    if [[ "$X" == TRUE ]]; then
19       P1=-6
20    fi
21    X=$(echo $IN | awk -F '|' '{print $3}')
22    if [[ "$X" == TRUE ]]; then
23       P1=-f
24    fi
25    sudo -k -S $SCRIPTPATH/$DEV.sh $CMD $C $P $P1 $P2<<!
26 $PASS
27 !
28 fi
```

Wenn man nicht mit Kommandozeilen hantieren will, is es möglich eine kleine graphische Oberfläche zu verwenden. Verwendet wird hier **yad**, eine bessere Alternative zu **Zenity**. **Yad** ist gegebenenfalls über das Paket-Managementsystem der jeweiligen Distribution zu installieren.

In Zeile 2 kann der Pfad zum Haupt-Script angegeben, wenn das Script sich in ein Verzeichnis wie */usr/local/bin* befindet, kann die Zeile „SCRIPTPATH=” lauten.

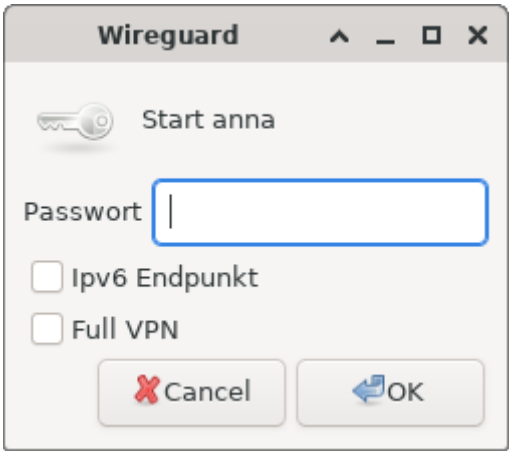
Zeile 3 und 4 sind von Interesse, wenn sich die Konfigurationsdateien sich nicht unter */etc/wireguard* befinden.

Die Maske wird abhängig, vom Vorhandensein der Tunnel Schnittstelle, (Prüfung in Zeile 6) vorgenommen.

Zeile 15 überprüft, ob die OK Schaltfläche betätigt wurde, wenn ja wird das Script weiter ausgeführt, sonst impliziert beendet.

Zeilen 25 bis 27 rufen mittels *sudo* unseren Hauptscript auf. Das Passwort wird über eine „here Dokument” eingelesen (*<<!* bis *!*).

4.7) Start/Stop GUI



Start Maske

5) Andere Plattformen

Das Verhalten und Eigenheiten der anderen Plattformen sollte auch kontrolliert werden.

5.1) Smartphone

Bei Smartphone sieht die Welt anders aus!

Scripting ist schwieriger, dafür kann definiert werden welche Applikationen über der VPN-Tunnel laufen. Wenn es nur um die Telefonie geht, ist es von Vorteil statische Adressen zu haben, die Wahl IPv4/IPv6 kann mittels 2 getrennte VPN Konfigurationsdateien erfolgen.

5.2) Windows

Ist es nicht ein Virus?

Die Frage kann eindeutig mit **ja** beantwortet werden. Durch illegale Vorgehensweise wurden Hersteller von PC/Notebook gezwungen exklusiv DOS und später Windows XX zu liefern. Der Kauf einen Rechner ohne Produkte von Microsoft stellt sich als extrem schwierig.

```
[Interface] # windows Site Zugriff
PrivateKey =  K...=
Address = 10.18.1.10/32
DNS = 192.168.178.2, fritz.box

[Peer] # Server
PublicKey  =  b...=
AllowedIPs = 10.18.1.0/24
EndPoint   = 192.0.2.1:51820
PersistentKeepAlive = 25
```

```
[Interface] # windows FULL-VPN
PrivateKey =  K...=
Address = 10.18.1.10/32, fd01:cafe::10/128
DNS = 192.168.178.2, fritz.box

[Peer] # Server
PublicKey  =  b...=
AllowedIPs = 0.0.0.0/0, ::/0
EndPoint   = 192.0.2.1:51820
PersistentKeepAlive = 25
```

Unterschiedlich sind nur der Zuweisung für **AllowedIPs**.
Im ersten Beispiel wird nur ein Adressenbereich, in dem Fall mit einer /24 Maske verwendet.
In der Zweite wird alles durchgelassen.

6) Server

6.1) Server

> Auf unserer Server ist vom Beginn an IPv4 und IPv6 sowie das Full-VPN Betrieb zu berücksichtigen.

6.2) Server Konfigurationsdatei

```
[Interface]
PrivateKey =  c...=
ListenPort = 51820
[Peer]
PublicKey  = v...=
AllowedIPs = 10.18.1.2/32, fd01:cafe::2/128, 0.0.0.0/0, ::/0
[Peer]
...
```

6.3) Server Script Konfiguration

```
CNF=/etc/wireguard/wg-server.var
WG=tun0
MTU=1420
IP4=10.18.1.1
IP6=fd01:cafe::1
IPR4=192.168.178.0/24
OIF=ens192
```

In den ersten Zeilen sind Werte, die direkt Wireguard betreffen, enthalten.

IPR4 steht für die Route unser Heim-Netz.

OIF bezeichnet die Netzwerkschnittstelle zur weiten Welt unseren Server.

6.4) Server Start Script

```
1  #!/usr/bin/bash
2  source /etc/wireguard/wg-server.var
3  case $1 in
4  start|up)
5      ip link add dev $WG type wireguard
6      wg setconf $WG $CNF
7      ip link set mtu $MTU up dev $WG
8      ip address add $IP6/64 dev $WG
9      ip address add $IP4/24 dev $WG
10     ip route add $IPR4/24 via $IP4 dev $WG
11     ipset create LAN nethash
12     ipset add LAN $IPR4
13     ipset add LAN $IP4/24
14     iptables -A FORWARD -m set ! --match-set LAN src -j ACCEPT
15     iptables -A INPUT -m set --match-set LAN src -j ACCEPT
16     iptables -t nat -A POSTROUTING -o $OIF -j MASQUERADE
17     ip6tables -A FORWARD -i $WG -j ACCEPT
18     ip6tables -t nat -A POSTROUTING -o $OIF -j MASQUERADE
19     ;;
20 stop|down)
21     ip link del dev $WG
22     ip6tables -D FORWARD -i $WG -j ACCEPT
23     ip6tables -t nat -D POSTROUTING -o $OIF -j MASQUERADE
24     iptables -t nat -A POSTROUTING -o $OIF -j MASQUERADE
25     iptables -D INPUT -m set --match-set LAN src -j ACCEPT
26     iptables -D FORWARD -m set ! --match-set LAN src -j ACCEPT
27     ipset destroy LAN
28     ;;
29 esac
```

Die Zeilen 5 bis 9 entsprechen die übliche Vorgehensweise beim Aufsetzen des Tunnels.

Zeile 10 stellt sicher, dass Adressen zum Heim-LAN über der VPN-Tunnel geleitet werden.

Ab Zeile 11 verwenden wir das Kommando **ipset**, das Paket muss installiert werden. Es wäre möglich gewesen etwas neuer zu verwenden, die Unterstützung bei Debian basierte Systeme ist nicht gegeben.

Mit **ipset** können einzelne Adressen oder Adressenbereiche definiert werden, damit reduziert sich den Aufwand an **iptables** Regeln.

Zeile 14 und 15 stellen sicher, dass je nach Adressenbereich das Netzwerkverkehr “genatet werden (14, 16) oder normal bearbeitet werden (15)

Für IPv6 ist nicht besonderes vorgesehen, es betrifft das gesamten IPv6 Netzwerkverkehr der “genatet” wird.

6.5) Systemd Unit Datei

```
[Unit]
Description=WireGuard Tunnel
After=network-online.target nss-lookup.target
Wants=network-online.target nss-lookup.target

[Service]
Type=oneshot
RemainAfterExit=yes
ExecStart=/usr/local/bin/wgs.sh up
ExecStop=/usr/local/bin/wgs.sh down
Environment=WG_ENDPOINT_RESOLUTION_RETRIES=infinity

[Install]
WantedBy=multi-user.target
```

7) CPE (Internet/WLAN Router)

> Die meist verwendeten Router dürften, die von AVM und Telekom sein.
> Jede Router hat seine eigenen Regeln.

7.1) Fritz!Box

Damit “fremde Adresse” z.b. 10.2.3.4 weitergereicht werden müssen statische Routen im Router gesetzt werden.

7.2) Speedport

Diese Geräte eignen sich scheinbar nicht für anspruchsvolle Betrieb.

7.3) Dlink, TP-Link. Netgear, …

Diese Geräte erlauben das Setzen von statische Routen, bei mancher nur für IPv4. Mit IPv6 kann es problematisch werden.

Eine Lösung wäre beispielsweise Openwrt auf die Geräte zu installieren. Damit erhält man ein Router der sein Name verdient.

Falls solch ein Router an einer DSL Buchse angeschlossen werden soll, dürfte der Zugang zum Internet problematisch sein.

8) Testumgebung

- Virtuelle Privat Server (VPS, bei Ionos) mit statische IPv4 und IPv6.
  - als Wireguard Server, Konfiguration wie hier beschrieben.
- Raspberry PI 4B.
  - als Gateway.
- NAS als DNS-Server für Subnetze und “Forwarding” auf der DSL Router.
- DSL Router Fritz!Box 7590.
- Freifunk Router.
  - für Windows Tests.

- Windows 10 (unter VirtualBox) auf der Hauptrechner.
- Notebook (Hauptrechner) mit Fedora 35.
    - Internet Anschluss über WLAN der Fritz!Box.
    - Für Windows Test Freifunk Router an der Ethernet Schnittstelle des Rechners.
        - Ethernet Schnittstelle nicht verwaltet.
- Weitere Notebook (Fedora 35), wahlweise über WLAN mit dem Heim-Netz oder Freifunk verbunden.

9) Test Ergebnisse

Vertrauen ist gut, Kontrolle ist besser.

9.1) Linux

Mit Linux kann alles auf ein Rechner laufen, die genaue Spezifizierung der Routen löst mögliche Probleme.

9.2) Windows

- Windows läuft in einer Virtuelle Maschine (VirtualBox).
    - 4 Konfigurationsdateien wurden verwendet.
        - wg4: Nur IPv4, Endpunkt über IPv4
        - wg6: Nur IPv4, Endpunkt über IPv6
        - wgf4: Full-VPN (IPv4/IPv6), Endpunkt IPv4
        - wgf6: Full-VPN (IPv4/IPv6), Endpunkt IPv6
    - Die Virtuelle Maschine hatte 2 Ethernet Schnittstelle
        - die Erste im Heim-LAN
        - die Zweite über FreiFunk

VPN zu Hause nach Hause über ein IPv6 Verbindung ist nicht die übliche Vorgehensweise, Windows war, hier ein wenig überfordert, mal ging es mal nicht.

9.3) Smartphone

Da das Smartphone nur für die Telefonie dienen soll, reicht der normale VPN-Betrieb aus, es sind gegebenenfalls zwei Konfigurationen zu verwenden (IPv4 Endpunkt / IPv4 Endpunkt).

9.4) MacOS

Mangel an passende Hardware konnte Tests nicht durchgeführt werden.
Es ist jedoch anzunehmen, dass es auf ein UNIXoid nicht viel anders ist, als mit Linux.

10) Zugriff auf SMB shares

Im Dateimanager werden keinen Shares angezeigt

Mit Eingabe vom **smb://nas/home** (Linux) oder **\\nas\home** (Windows) lässt sich der Zugriff erreichen. Damit ist ein Layer-3 Tunnel, wie von Wireguard verwendet, nicht wirklich ein Nachteil.