WireGuard

IT Security Workshop

Sascha Turban, Tobias Flaig

12. Oktober 2018

Motivation

- IPsec: komplex, sichere Konfiguration schwierig
- OpenVPN: langsam, da teilweise im User Space
- bisherige VPNs: viele LOC, Verifikation schwierig

LOC verschiedener VPN-Lösungen[1]

[1] Jason A Donenfeld. *WireGuard. Fast, Modern, Secure VPN Tunnel*. 2018. URL: https://www.wireguard.com/talks/blackhat2018-slides.pdf (besucht am 08. 10. 2018).

- WireGuard soll eine Alternative bieten
  - Sicher
  - Performant
  - Einfache Bedienung

Umsetzung

- Layer 3 (und nur Layer 3)
- Virtuelles Kernel-Netzwerkinterface für Linux
- Identitäten an Schlüssel gebunden
- Einfachheit in der Nutzung und des Designs
- Statische Header fester Länge
- Statische Speicherallokation
- Stealth
- Starke Kryptografie
- Robustheit gegen Angriffe

- Layer 3 (und nur Layer 3)
- Virtuelles Kernel-Netzwerkinterface für Linux
- Identitäten an Schlüssel gebunden
- Einfachheit in der Nutzung und des Designs
  - Für den Nutzer sichtbar: Netzwerkschnittstelle
  - Zusammenarbeit mit Standardtools (z.B. iptables)
- Statische Header fester Länge
- Statische Speicherallokation
- Stealth
- Starke Kryptografie
- Robustheit gegen Angriffe

- Layer 3 (und nur Layer 3)
- Virtuelles Kernel-Netzwerkinterface für Linux
- Identitäten an Schlüssel gebunden
- Einfachheit in der Nutzung und des Designs
- Statische Header fester Länge
  - Vorteil: Sicherer, schneller, kein dynamischer Speicher
  - Nachteil: keine abwärtskompatible Erweiterbarkeit
- Statische Speicherallokation
- Stealth
- Starke Kryptografie
- Robustheit gegen Angriffe

- Layer 3 (und nur Layer 3)
- Virtuelles Kernel-Netzwerkinterface für Linux
- Identitäten an Schlüssel gebunden
- Einfachheit in der Nutzung und des Designs
- Statische Header fester Länge
- Statische Speicherallokation
- Stealth
- Starke Kryptografie
- Robustheit gegen Angriffe

- Reduktion der Angriffsfläche beim Design
- Nutzung moderner kryptografischer Direktiven
- Einheitliche/Restriktive Konfiguration & Transparentes Verhalten
- Sauberes Design

- Reduktion der Angriffsfläche beim Design
  - Dynamische Speicherallokation wird vermieden
  - Keine Speicherallokation für nicht authentifizierte Kommunikation
  - Replay attacks
  - Amplification attacks
  - CPU-Exhaustion Attacks
  - Stealth
- Nutzung moderner kryptografischer Direktiven
- Einheitliche/Restriktive Konfiguration & Transparentes Verhalten
- Sauberes Design

- Reduktion der Angriffsfläche beim Design
- Nutzung moderner kryptografischer Direktiven
- Einheitliche/Restriktive Konfiguration & Transparentes Verhalten
- Sauberes Design

- Reduktion der Angriffsfläche beim Design
- Nutzung moderner kryptografischer Direktiven
- Einheitliche/Restriktive Konfiguration & Transparentes Verhalten
- Sauberes Design
  - Absicherung des Protokolldesigns mit formaler Verifikation
  - Verständliche Implementierung vor Schichtenarchitektur
  - Kleine Codebasis
  - "Do one thing and do it well"

- Kryptografische Funktionen
  - Symmetrische Verschlüsselung: ChaCha20Poly1305 AEAD
  - Asymmetrische Verschlüsselung: Curve25519 ECDH
  - Hashfunktion: Blake2s
    - MAC
    - HMAC
    - HKDF
- Eigene Implementierung dieser Funktionen
  - Kernel Krypto API wird nicht genutzt

type := 0x1 (1 byte)	reserved := 0^3 (3 bytes)
sender := I_i (4 bytes)	
ephemeral (32 bytes)	
static ($\widehat{32}$ bytes)	
timestamp ($\widehat{12}$ bytes)	
mac1 (16 bytes)	mac2 (16 bytes)

Initial message

type := 0x2 (1 byte)	reserved := 0^3 (3 bytes)
sender := I_r (4 bytes)	receiver := I_i (4 bytes)
ephemeral (32 bytes)	
empty ($\widehat{0}$ bytes)	
mac1 (16 bytes)	mac2 (16 bytes)

Response message

type := 0x3 (1 byte)	reserved := 0^3 (3 bytes)
receiver := $I_{m'}$ (4 bytes)	
nonce := ρ^{24} (24 bytes)	
cookie ($\widehat{16}$ bytes)	

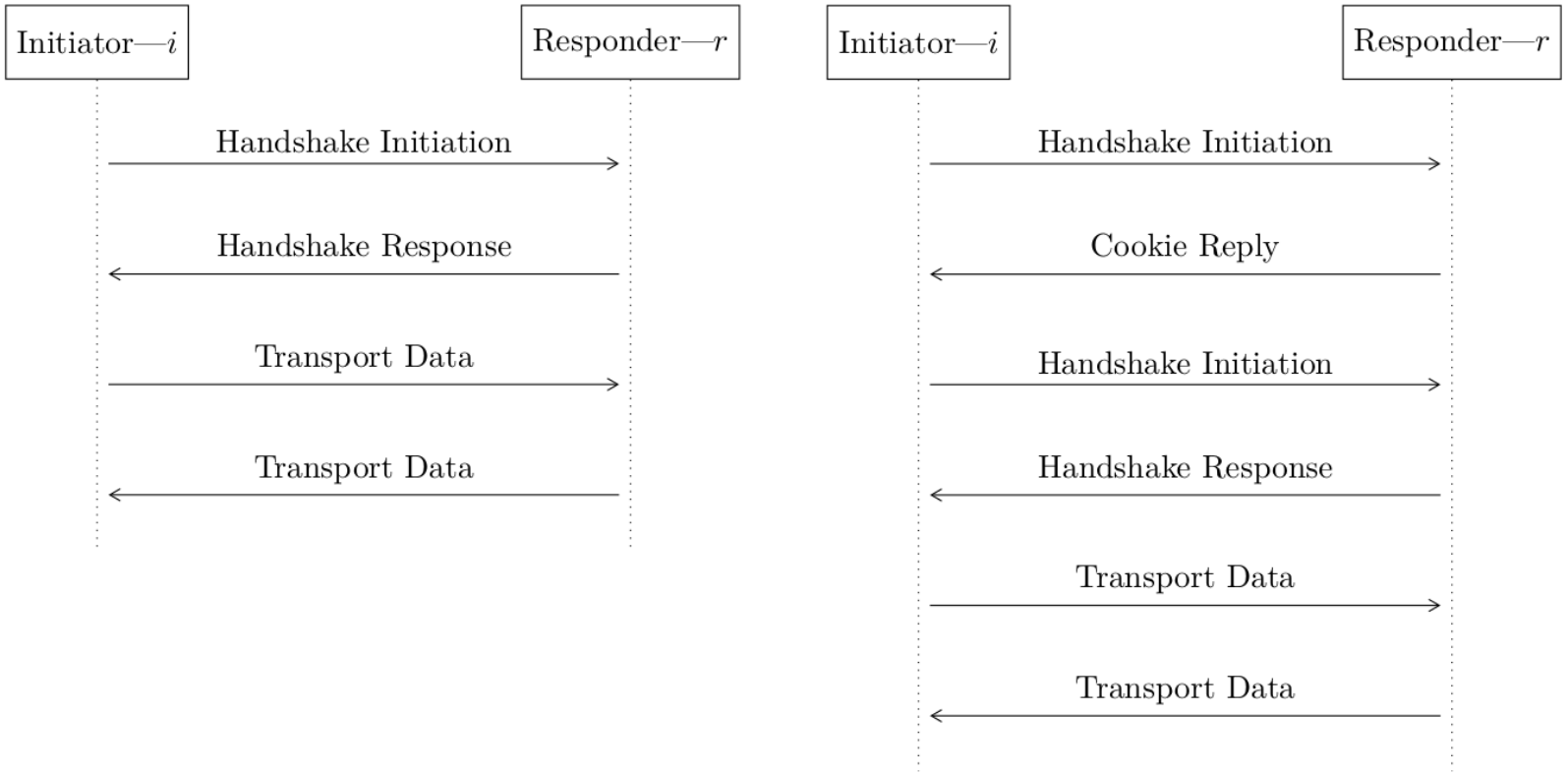
Cookie message

type := 0x4 (1 byte)	reserved := 0^3 (3 bytes)
receiver := $I_{m'}$ (4 bytes)	
counter (8 bytes)	
packet ($\widehat{\|P\|}$ bytes)	

Transport message

Nachrichtentypen[2]

[2] Jason A Donenfeld. „WireGuard: Next Generation Kernel Network Tunnel.“. In: *NDSS*. 2017.



Kommunikation zwischen zwei Peers[3]

[3] Donenfeld, „WireGuard: Next Generation Kernel Network Tunnel.“

Handshake

- 1-RTT key exchange mittels Curve25519 ECDH
- Bei Überlast wird ein Kryptografischer Cookie ausgetauscht
- Handshake wird spätestens alle 3 Minuten wiederholt

type := 0x1 (1 byte)	reserved := 0^3 (3 bytes)
sender := I_i (4 bytes)	
ephemeral (32 bytes)	
static ($\widehat{32}$ bytes)	
timestamp ($\widehat{12}$ bytes)	
mac1 (16 bytes)	mac2 (16 bytes)

Initial message[4]

[4] Donenfeld, „WireGuard: Next Generation Kernel Network Tunnel.“

Cookie

Wird vom Responder zum Initiator gesendet, falls der Responder unter Überlast leidet

- Was bringts?
- Was kostet das?
- Wo liegen Probleme?

Wird vom Responder zum Initiator gesendet, falls der Responder unter Überlast leidet

- Was bringts?
  - Verhindert CPU-exhaustion Angriffe
- Was kostet das?
- Wo liegen Probleme?

Wird vom Responder zum Initiator gesendet, falls der Responder unter Überlast leidet

- Was bringts?
- Was kostet das?
  - Erhöht RTT des Handshakes
- Wo liegen Probleme?

Cookie

Wird vom Responder zum Initiator gesendet, falls der Responder unter Überlast leidet

- Was bringts?
- Was kostet das?
- Wo liegen Probleme?
  - Soll Stealth Eigenschaft nicht zerstören
  - MITM soll mit abgefangenen Cookies keine Authentifizierung gelingen
  - Soll auch funktionieren wenn sowohl Initiator als auch Responder überlastet sind

Cookie

Wie erreichen wir das?

- MAC1 & MAC2 in jeder Nachricht
- MAC1 := MAC(Pubkey$_R$, Message)
- MAC1 wird vor Generierung des Cookies geprüft
- Cookie:= AEAD(Pubkey$_I$, MAC1 || RandomValue) basierend auf MAC1 des Initiators
- MAC2:=MAC(Cookie, Message)

Zusammenfassung

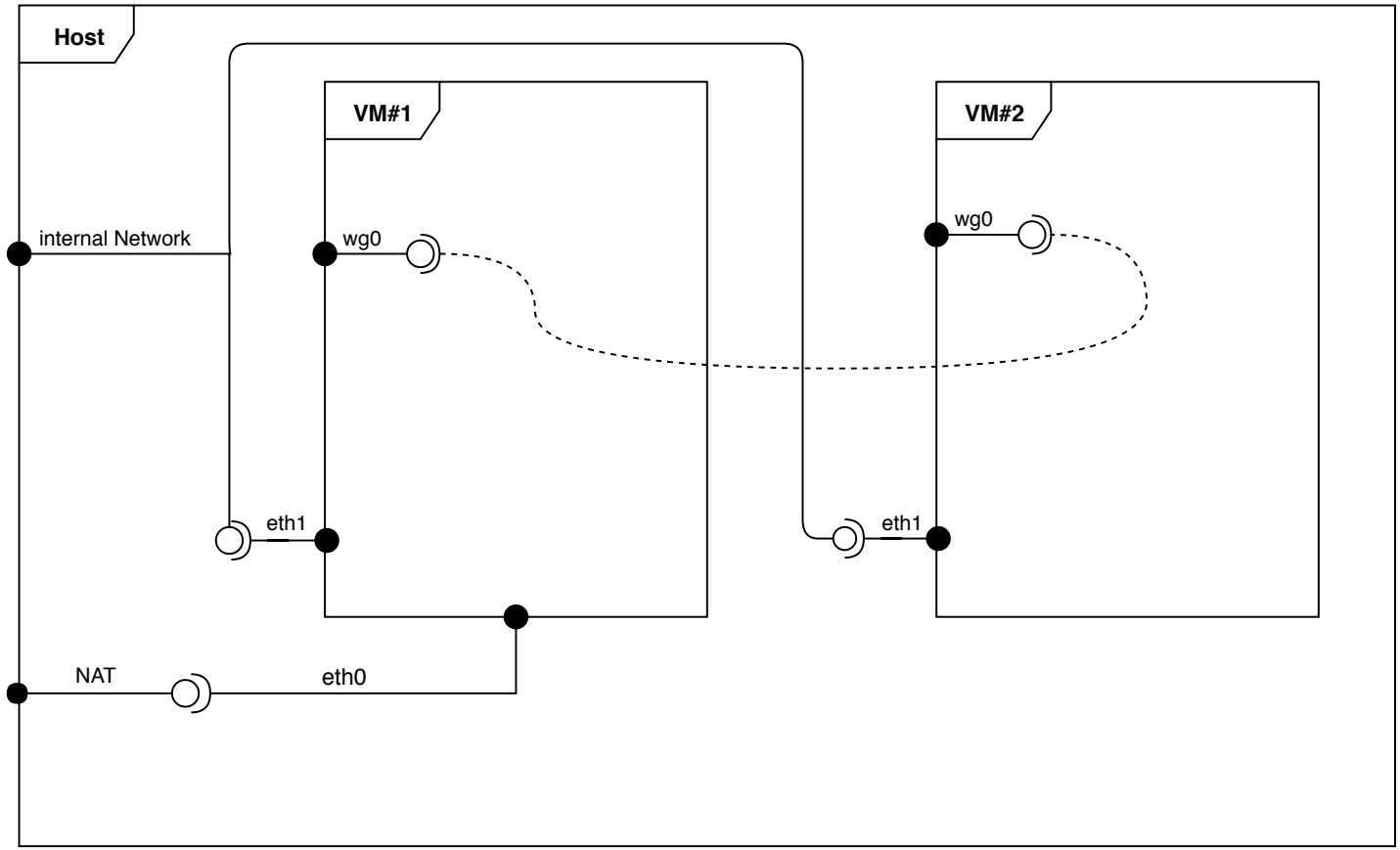
- MAC1 ermöglicht schnelle, schwache Authentifizierung
  - MAC1 kann von Angreifern errechnet werden, die über die öffentlichen Schlüssel der Clients verfügen
- Cookie kann vom Initiator geprüft werden und dient zur Berechnung von MAC2
- MAC2 kann vom Responder geprüft werden und ermöglicht anschließend einen normalen Handshake

Anwendung

- 2 VMs mit Debian 9 (VirtualBox)

- 2 VMs mit Debian 9 (VirtualBox)
- 2 Netzwerkschnittstellen auf VM1
  - NAT für Internetzugriff
  - Internes Netzwerk für Kommunikation mit Host und VM2

- 2 VMs mit Debian 9 (VirtualBox)
- 2 Netzwerkschnittstellen auf VM1
  - NAT für Internetzugriff
  - Internes Netzwerk für Kommunikation mit Host und VM2
- VM2 mit internem Netzwerk verbunden
- Ziel: VM2 soll über VM1 mit dem Internet kommunizieren



Schematischer Versuchsaufbau

- Schlüssel generieren
  - `wg genkey | tee wg-private.key | wg pubkey > wg-public.key`

- Schlüssel generieren
  - `wg genkey | tee wg-private.key | wg pubkey > wg-public.key`
- `/etc/wireguard/wg0.conf`
  - WireGuard-Konfigurationsdatei

- Schlüssel generieren
  - `wg genkey | tee wg-private.key | wg pubkey > wg-public.key`
- `/etc/wireguard/wg0.conf`
  - WireGuard-Konfigurationsdatei
- `/etc/sysctl.d/99-sysctl.conf`
  - `net.ipv4.ip_forward = 1`

- Schlüssel generieren
  - `wg genkey | tee wg-private.key | wg pubkey > wg-public.key`
- `/etc/wireguard/wg0.conf`
  - WireGuard-Konfigurationsdatei
- `/etc/sysctl.d/99-sysctl.conf`
  - `net.ipv4.ip_forward = 1`
- Starten und Beenden von WireGuard
  - `systemctl start wg-quick@wg0`
  - `systemctl stop wg-quick@wg0`

- Schlüssel generieren
  - `wg genkey | tee wg-private.key | wg pubkey > wg-public.key`
- `/etc/wireguard/wg0.conf`
  - WireGuard-Konfigurationsdatei
- `/etc/sysctl.d/99-sysctl.conf`
  - `net.ipv4.ip_forward = 1`
- Starten und Beenden von WireGuard
  - `systemctl start wg-quick@wg0`
  - `systemctl stop wg-quick@wg0`
- Verbundene Clients anzeigen
  - `wg`

```
[Interface]
PrivateKey = ...
ListenPort = 5555
Address = 10.0.0.1/24
PostUp = iptables -A FORWARD -i %i -j ACCEPT; \
    iptables -A FORWARD -o %i -j ACCEPT; \
    iptables -t nat -A POSTROUTING -o enp0s3 \
    -j MASQUERADE
PostDown = iptables -D FORWARD -i %i -j ACCEPT; \
    iptables -D FORWARD -o %i -j ACCEPT; \
    iptables -t nat -D POSTROUTING -o enp0s3 \
    -j MASQUERADE

[Peer]
PublicKey = ...
AllowedIPs = 10.0.0.2/32
```

Konfigurationsdatei auf VM2 (Client)


```
[Interface]
PrivateKey = KKit50ILjssF39X7OuXpYP8oc6szZrXXTO3ZTvLcfk0=
ListenPort = 5555
Address = 10.0.0.2/24

[Peer]
PublicKey = o6LaXgzfHH5ryKR7iOTiMdhrPtmpbdgP8fxVeOmpOwY=
AllowedIPs = 0.0.0.0/0, ::0/0
Endpoint = 192.168.56.101:5555
PersistentKeepalive = 25
```

- /etc/network/interfaces

```
auto wg0
iface wg0 inet static
    address 10.0.0.1
    netmask 255.255.255.0
    pre-up ip link add $IFACE type wireguard
    pre-up wg setconf $IFACE /etc/wireguard/$IFACE.conf
    post-down ip link del $IFACE
iface wg0 inet6 static
    address 2001:db8:1234:5678::1
    netmask 64
```

[5] *Wireguard - Debian Wiki*. 2018. URL: https://wiki.debian.org/Wireguard (besucht am 08. 10. 2018).

- /etc/systemd/network/wg0.netdev

```
[NetDev]
Name=wg0
Kind=wireguard
Description=Wireguard test

[WireGuard]
PrivateKey=...
ListenPort=5555

[WireGuardPeer]
PublicKey=...
AllowedIPs=0.0.0.0/0
AllowedIPs=::/0
Endpoint=<remote IP or hostname>:<remote port>
```

[6] *Wireguard - Debian Wiki*.

OpenVPN Server-Beispielkonfiguration (1)


```
# local IP and port to listen on
;local a.b.c.d
port 1194

# tcp or udp configuration
proto udp

# "dev tun" will create a routed IP tunnel,
# "dev tap" will create an ethernet tunnel.
dev tun

# TAP-Win32 adapter name (Windows only)
;dev-node MyTap

# SSL/TLS root cert (ca), cert and private key
ca ca.crt
cert server.crt
key server.key  # This file should be kept secret
```

OpenVPN Server-Beispielkonfiguration (2)


```
# Diffie hellman parameters.
dh dh2048.pem

# Network topology
;topology subnet

# VPN subnet (not used for Ethernet bridging)
server 10.8.0.0 255.255.255.0

# client <-> virtual IP address associations
ifconfig-pool-persist ipp.txt

# Configure server mode for ethernet bridging
;server-bridge 10.8.0.4 255.255.255.0 10.8.0.50 10.8.0.100

# Push routes to the client
;push "route 192.168.10.0 255.255.255.0"
;push "route 192.168.20.0 255.255.255.0"
```

OpenVPN Server-Beispielkonfiguration (3)


```
# Access a subnet behind a client
;client-config-dir ccd
;route 192.168.40.128 255.255.255.248

# Give a fixed VPN IP address to a client
;client-config-dir ccd
;route 10.9.0.0 255.255.255.252

# different firewall access policies for clients
;learn-address ./script

# configure clients to redirect their default
# network gateway through the VPN
;push "redirect-gateway def1 bypass-dhcp"

# Push Windows-specific network settings to clients
;push "dhcp-option DNS 208.67.222.222"
;push "dhcp-option DNS 208.67.220.220"
```

OpenVPN Server-Beispielkonfiguration (4)


```
# allow different clients to see each other
;client-to-client

# allow clients to connect with the same cert/key/CN
;duplicate-cn

# ping every 10s, assume client is down after 120s
keepalive 10 120

# HMAC firewall
tls-auth ta.key 0 # This file is secret

# Cryptographic cipher
cipher AES-256-CBC

# Enable compression on the VPN link
;compress lz4-v2
;push "compress lz4-v2"
```

OpenVPN Server-Beispielkonfiguration (5)


```
# Compatibility with older clients
;comp-lzo

# max number of concurrently connected clients
;max-clients 100

# reduce OpenVPN daemon's privileges
;user nobody
;group nobody

# try to avoid accessing certain resources on restart
persist-key
persist-tun

# short status file
status openvpn-status.log
```

OpenVPN Server-Beispielkonfiguration (6)


```
# extra log for OpenVPN messages
;log         openvpn.log
;log-append  openvpn.log

# log file verbosity
verb 3

# Silence repeating messages
;mute 20

# Notify the client that when the server restarts
explicit-exit-notify 1
```

OpenVPN Client-Beispielkonfiguration (1)


```
# Specify that we are the client
client

# Same setting as on the server
dev tun

# TAP-Win32 adapter name (Windows only)
;dev-node MyTap

# tcp or udp configuration
proto udp

# hostname/IP and port of the server(s)
remote my-server-1 1194
;remote my-server-2 1194

# random host for load balancing
;remote-random
```

OpenVPN Client-Beispielkonfiguration (2)


```
# try indefinitely to resolve server hostname
resolv-retry infinite

# no binding to specific local port
nobind

# Downgrade privileges after initialization
;user nobody
;group nobody

# Try to preserve some state across restarts.
persist-key
persist-tun

# connect through HTTP proxy
;http-proxy-retry # retry on connection failures
;http-proxy [proxy server] [proxy port #]
```

OpenVPN Client-Beispielkonfiguration (3)


```
# silence duplicate packet warnings
;mute-replay-warnings

# SSL/TLS parms
ca ca.crt
cert client.crt
key client.key

# Verify server certificate
remote-cert-tls server

# If a tls-auth key is used on the server
tls-auth ta.key 1

# Select a cryptographic cipher
cipher AES-256-CBC
```

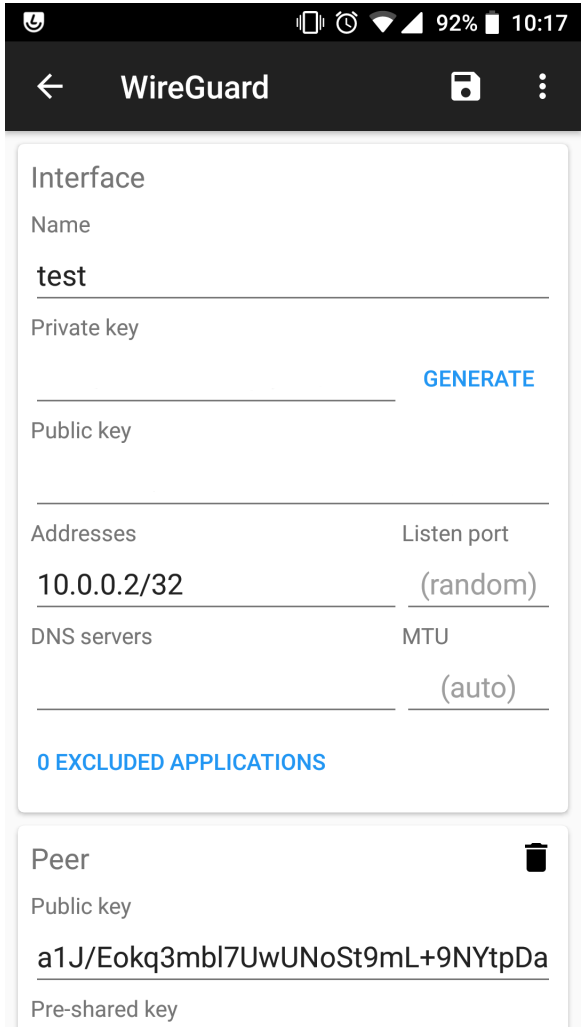
OpenVPN Client-Beispielkonfiguration (4)

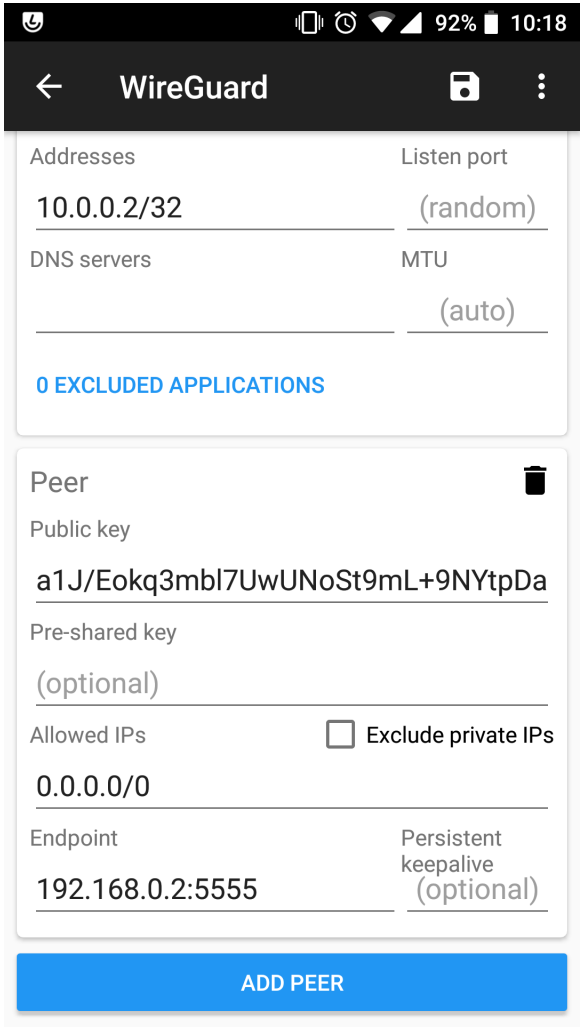

```
# Enable compression on the VPN link
;comp-lzo

# Set log file verbosity
verb 3

# Silence repeating messages
;mute 20
```

Unterstützte Betriebssysteme[7]

- GNU/Linux
- OpenWRT
- FreeBSD, OpenBSD
- macOS
- Android
- Windows: “coming soon”
  - Userspace-Implementierung geplant
  - TunSafe: proprietäre Implementierung

[7] *Installation - WireGuard*. 2018. URL: https://www.wireguard.com/install/ (besucht am 05. 10. 2018).





Android Client

- Konfigurationsdatei mit Private Key notwendig
  - Kann für Android-Client auch mit QR-Code eingelesen werden

- Konfigurationsdatei mit Private Key notwendig
  - Kann für Android-Client auch mit QR-Code eingelesen werden
- Vom Server generieren und dem Endnutzer anbieten?
  - Shell-Script: siehe SAR-Wiki

- Konfigurationsdatei mit Private Key notwendig
  - Kann für Android-Client auch mit QR-Code eingelesen werden
- Vom Server generieren und dem Endnutzer anbieten?
  - Shell-Script: siehe SAR-Wiki
- Besser: Lokale Generierung
  - Erfordert aber manuelle Konfiguration auf dem Client
  - Optimal?

Demo

Fazit

Fazit

- „Blasphemische“ Abkehr von etablierten Standards
- Methodisches Design
- Versprechen werden in vielen Bereichen gehalten

Fazit

- „Blasphemische“ Abkehr von etablierten Standards
  - Krypto: ASN1, X.509
  - ISO/OSI Schichtenarchitektur wird aufgebrochen
  - Keine „cipher agility“
- Methodisches Design
- Versprechen werden in vielen Bereichen gehalten

- „Blasphemische“ Abkehr von etablierten Standards
- Methodisches Design
  - Zustandsautomat für Wireguard Protokoll
  - Formale Verifikation des Protokolls
  - Bekannte Sicherheitsrisiken wurden gezielt vermieden
- Versprechen werden in vielen Bereichen gehalten

- „Blasphemische“ Abkehr von etablierten Standards
- Methodisches Design
  - Zustandsautomat für Wireguard Protokoll
  - Formale Verifikation des Protokolls
  - Bekannte Sicherheitsrisiken wurden gezielt vermieden
    - Keine dynamische Speicherallokation
    - Keine Parser, statische Felder
    - Vorkehrungen gegen DOS Angriffe
- Versprechen werden in vielen Bereichen gehalten

- „Blasphemische“ Abkehr von etablierten Standards
- Methodisches Design
- Versprechen werden in vielen Bereichen gehalten
  - Einfachheit in der Nutzung und im Aufbau
    - Niedige #LOC
    - Für die Konfiguration sind nur öffentlich Schlüssel und Kenntnis selbst gewählter IP-Adressen nötig
  - Performance

- Aber was ist mit...
  - ...Einbindung einer PKI?
  - ...verschiedenen Programmversionen?
- WireGuard wurde bereits für die Aufnahme in den Linuxkernel vorgeschlagen mit einigen prominenten Unterstützern des Vorschlags u.a. Linus Torvalds

Diskussion und Fragen

Donenfeld, Jason A. *WireGuard. Fast, Modern, Secure VPN Tunnel*. 2018. URL: https://www.wireguard.com/talks/blackhat2018-slides.pdf (besucht am 08. 10. 2018).

— „WireGuard: Next Generation Kernel Network Tunnel.“. In: *NDSS*. 2017.

Installation - WireGuard. 2018. URL: https://www.wireguard.com/install/ (besucht am 05. 10. 2018).

Wireguard - Debian Wiki. 2018. URL: https://wiki.debian.org/Wireguard (besucht am 08. 10. 2018).