WireGuard ou comment mettre en place un VPN en un temps record

Vincent Autefage, Damien Magoni

WireGuard ou comment mettre en place un VPN en un temps record

Vincent Autefage
IUT Informatique - Université de Bordeaux
15 Rue Naudet
33175 Gradignan Cedex, France

Damien Magoni
LaBRI - Université de Bordeaux
351 Cours de la Libération
33405 Talence Cedex, France

Résumé

Les VPNs représentent une technologie de plus en plus populaire notamment avec la généralisation du travail à distance. En effet, l'accès aux ressources d'un réseau privé depuis l'extérieur est devenu un élément critique de sécurité mais aussi de continuité opérationnelle et pédagogique.

Bien qu'un certain nombre de solutions Open Source de serveurs VPN soient disponibles depuis de nombreuses années, WireGuard présente une philosophie radicalement différente de la concurrence. Il dispose d'une implémentation légère ayant la particularité d'être intégrée comme module au noyau Linux depuis 2020. WireGuard ne se limite néanmoins pas au monde Linux ; il est en effet disponible sur un grand nombre de plateformes (e.g. BSD, Windows, Mac OSX, Android, iOS). Son concepteur ayant souhaité son utilisation la plus simple possible, il s'est grandement inspiré du modèle des clés SSH rendant ainsi la mise en place de WireGuard des plus triviales.

Cet article présente les concepts principaux régissant l'architecture de WireGuard (e.g. réseau, cryptographie) ainsi qu'une comparaison de performance avec des projets plus classiques tels que OpenVPN. Nous présentons enfin des cas d'usage réels pour lesquels une telle solution nous paraît pertinente.

Mots-clefs

Cryptographie, OpenVPN, Réseaux Privés Virtuels, VPN, WireGuard

1 Introduction

Un réseau privé virtuel (*i.e.* VPN pour *Virtual Private Network*) permet d'étendre un réseau privé en traversant un réseau public et permet donc à ses utilisateurs d'envoyer et de recevoir des données à travers ce dernier comme si leurs équipements étaient directement connectés au réseau privé. Leur émergence date du milieu des années 90.

Les VPNs représentent une technologie de plus en plus populaire notamment avec la généralisation du travail à distance [01]. En effet, l'accès aux ressources d'un réseau privé depuis l'extérieur est devenu un élément critique de sécurité mais aussi de continuité opérationnelle et pédagogique.

Le chiffrement des données échangées est fréquent bien qu'il ne soit pas requis. Un VPN est construit en établissant une ou plusieurs connexions point-à-point virtuelles à l'aide de circuits dédiés ou de protocoles permettant la création de tunnels au travers des réseaux existants.

WireGuard [02] est un VPN développé depuis 2016 proposant une implémentation concise (*i.e.* moins de 4000 lignes de code) ayant la particularité d'être intégrée comme module du noyau Linux depuis le début de 2020. *WireGuard* a également été porté sous forme applicative sur une grande variété de systèmes (*e.g.* Windows, Android, MacOS X, iOS, BSD dont pfSense et OPNSense) ; une version expérimentale appelée *WireGuard NT* implémente une version noyau pour les systèmes Windows. *WireGuard* se définit selon ses propres termes comme plus simples d'utilisation, plus performant que ses concurrents, tout en proposant une surface d'attaque minimale.

Dans un premier temps, nous proposons un état de l'art des solutions VPNs libres les plus courantes afin de mettre en lumière les principaux éléments sur lesquels est construit *WireGuard*. Nous nous intéressons notamment aux choix cryptographiques faits par son auteur. Par la suite nous présentons des résultats d'expérimentations afin d'évaluer les performances promises par cette solution. Enfin, nous décrivons l'utilisation de l'outil ainsi que deux cas d'usage réels.

2 WireGuard dans le paysage des VPNs Open Source

Il existe de nombreux types de réseaux virtuels qui peuvent être classés par leur topologie ainsi que par leur technologie comme illustrés sur la figure 1 [03].

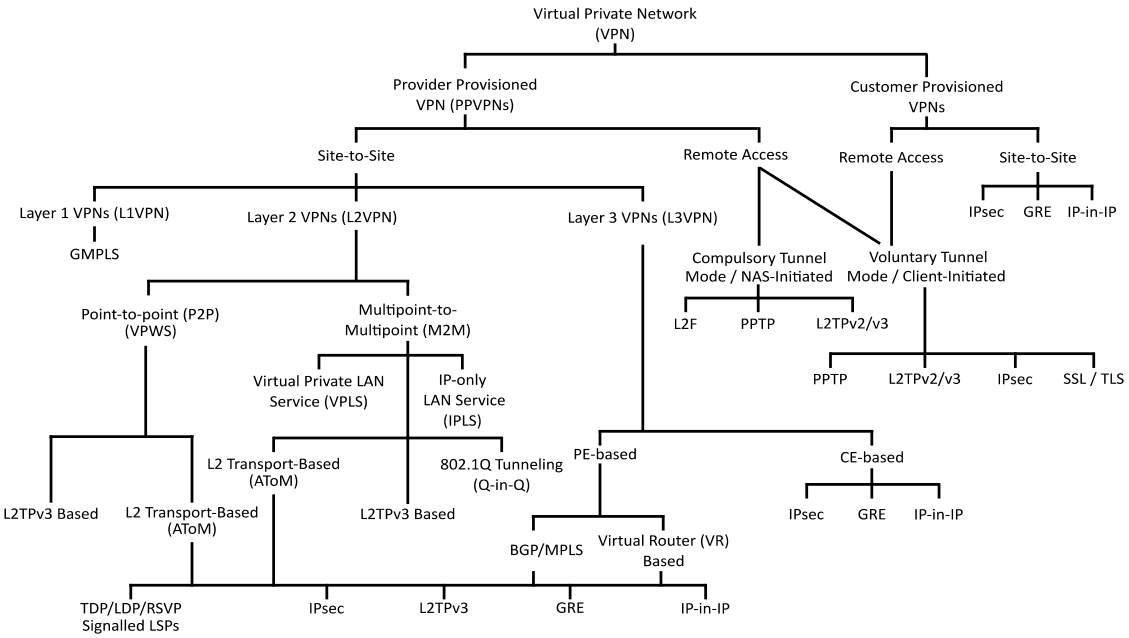

Figure 1 – Classification des VPN

Lorsqu'un VPN est créé en s'appuyant sur un réseau public tel qu'Internet, il met alors en œuvre de l'authentification et du chiffrement afin de garantir la sécurité des communications. Les deux solutions les plus fréquemment utilisées dans ce cas de figure sont *IPsec* et *TLS*.

IPsec [04] permet de sécuriser le trafic au niveau de la couche réseau du modèle OSI. Il se compose de trois protocoles : *ESP* (authentification et chiffrement des données), *AH* (authentification de l'entête et des données) et *IKE* (gestion des associations et des échanges de clés). Il possède un mode *tunnel* et un mode *transport*. Le mode *tunnel* sert à interconnecter deux sites distants et encapsule les paquets IP du réseau privé dans des paquets IP du réseau public. Le mode *transport* quant à lui n'effectue pas d'encapsulation IP. *IPsec* est une suite de protocoles assez complexes à mettre en œuvre. Il peut être implémenté aussi bien dans du matériel dédié (*e.g.* Cisco ASA) que sur une machine standard. La partie authentification et chiffrement est alors effectuée dans le noyau du système d'exploitation alors que la partie échange de clés s'effectue dans l'espace utilisateur. Il existe plusieurs implémentations open-source qui tournent sur de telles machines (*e.g. Openswan* [05] et *StrongSwan* [06]). Son utilisation est plus généralement observée en entreprise au regard de sa complexité de mise en place.

TLS [07] permet de sécuriser le trafic au niveau de la couche transport du modèle OSI. Il s'appuie sur le protocole TCP. *DTLS* est est une version modifiée de *TLS* pour le protocole UDP. Ces deux versions permettent de sécuriser non plus toute une partie du trafic IP d'une machine mais des flux spécifiques de données créés par des *sockets*. Sa granularité est donc plus fine que celle d'*IPsec*. *TLS* est la technologie sur laquelle repose le protocole HTTPS qui sécurise les connexions sur le web. Il peut être utilisé pour former des VPNs en encapsulant le trafic au sein d'une ou plusieurs connexions sécurisées. Il est en général implémenté sur des machines standards. Son implémentation libre la plus connue est *OpenV*PN [08] qui peut utiliser *TLS* ou *DTLS* et fournit à la fois le serveur et le client. *OpenVPN* est en général utilisé par les particuliers, les petites entreprises ou encore les solutions d'accès au Cloud. Ses performances sont parfois considérées comme sous-optimales car son implémentation s'exécute dans l'espace utilisateur.

WireGuard [09] est un protocole relativement récent dont le développement a débuté en 2016. Il permet de créer des VPNs sécurisés par cryptographie tout comme *IPsec* et *OpenVPN*. D'un point de vue architectural, il est plus proche d'*IPsec* car il se place lui aussi au niveau de la couche réseau. Sa plus grande particularité est d'être intégralement intégré au noyau Linux depuis 2020. C'est un protocole minimaliste qui impose un certain nombre de contraintes afin d'augmenter sa sécurité et son efficacité. Tout d'abord il s'appuie uniquement sur UDP pour encapsuler le trafic chiffré. En outre, il n'utilise qu'une unique suite d'algorithmes cryptographiques (*i.e.* un ensemble d'algorithmes ayant chacun une fonction spécifique) :

- Curve25519 pour l'échange de clés,
- ChaCha20 pour le chiffrement symétrique,
- Poly1305 pour les codes d'authentification de messages,
- SipHash pour les clés de hachage,
- BLAKE2s pour la fonction de hachage cryptographique.

A contrario, *TLS* dans se version 1.2 propose 37 suites différentes, sa version 1.3 n'en propose en revanche que 5 [12]. *IPsec* propose quant à lui une trentaine d'algorithmes de chiffrement, une douzaine d'algorithmes d'intégrité et une vingtaine d'algorithmes de génération de clés [13] pouvant être combinés pour former une multitude de suites cryptographiques. *WireGuard* ne prend

en charge ni la gestion des clés statiques, ni des certificats, ni des standards d'infrastructure à clés publiques (*i.e.* PKI pour *Public Key Infrastructure*, *e.g.* X509, ASN.1). C'est donc à l'utilisateur de fournir les clés asymétriques nécessaires à la sécurisation des échanges. *WireGuard* peut ainsi être comparé au modèle de clés utilisées pour sécuriser une connexion SSH.

Le résultat de ces choix et contraintes techniques est une implémentation dont la taille est d'environ 4000 lignes de code. Elle est donc facilement vérifiable contrairement à celles de S*trongSwan* ou d'*OpenVPN* dont les implémentations dépassent les 400 000 lignes. Plusieurs preuves formelles et automatiques de *WireGuard* ont ainsi pu être publiées [14][15].

3 Expérimentations

WireGuard est souvent mis en avant pour la rapidité de ses transferts mais également pour sa faible latence. Les résultats présentés dans les figures 2 et 3 sont extraites du site officiel de *WireGuard* [02].

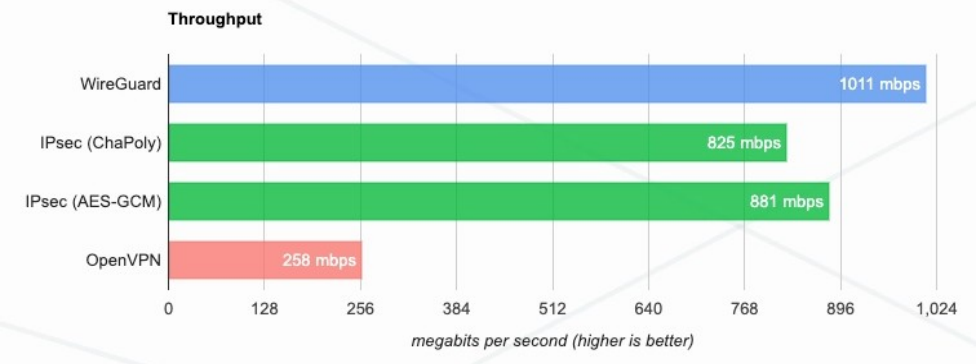

Figure 2 – Résultats de débit donnés par WireGuard

Figure 3 – Résultats de latence donnés par WireGuard

Ces résultats expérimentaux obtenus en 2017 mettent en avant des performances accrues pour *WireGuard* ainsi que pour *IPsec* à contrario d'*OpenVPN* qui est ici configuré avec UDP/DTLS en version 1.2. De notre point de vue, la raison principale de cet écart est la probable absence d'accélération matérielle pour les opération cryptographique symétrique pour *OpenVPN* dans cette expérimentation. Les processeurs modernes sont en effet équipés d'unités de calculs dédiées permettant d'effectuer les opérations de chiffrement et déchiffrement pour certains protocoles de cryptographie symétrique. Dans le cas d'*IPSec* et d'*OpenVPN*, le protocole AES [16] est le plus

souvent utilisé. *WireGuard* repose quant à lui sur ChaCha20 [17] qui peut lui aussi être accéléré matériellement. Les faibles performances d'*OpenVPN* dans ces résultats nous poussent à croire que l'accélération matérielle n'a pas été activée pour cet outil dans le cadre de ces mesures.

Afin de vérifier les gains actuels de performances proposés par *WireGuard* sur *OpenVPN* (en TCP/TLS et UDP/DTLS en version 1.3), nous avons mené des expérimentations similaires à celles proposées sur le site de *WireGuard* en conditions réelles avec accélération matérielle ainsi qu'au sein d'un réseau virtuel sans accélération matérielle cryptographique.

La première expérimentation a été effectuée avec 2 machines physiques tournant sous *Debian 11* avec un noyau *Linux 5.10* équipées d'un *Intel Core i7-9750H*. Les deux machines sont connectées en gigabit au travers d'un commutateur gigabit dédié. Nous avons effectué des tests de débit ainsi que de latence avec les outils iperf et iperf3 [10] ; outils également utilisés pour obtenir les résultats présentés précédemment. Les résultats de ces tests son exposés en figures 4 et 5.

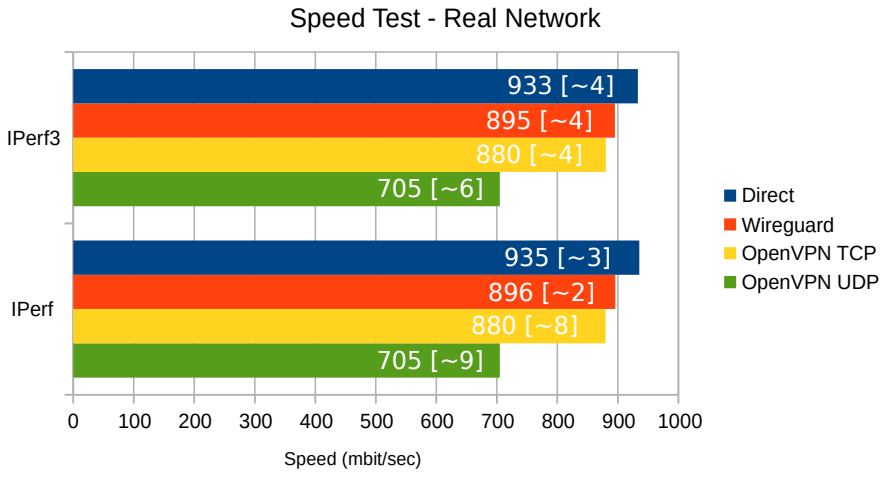

Figure 4 – Débit en réseau réel

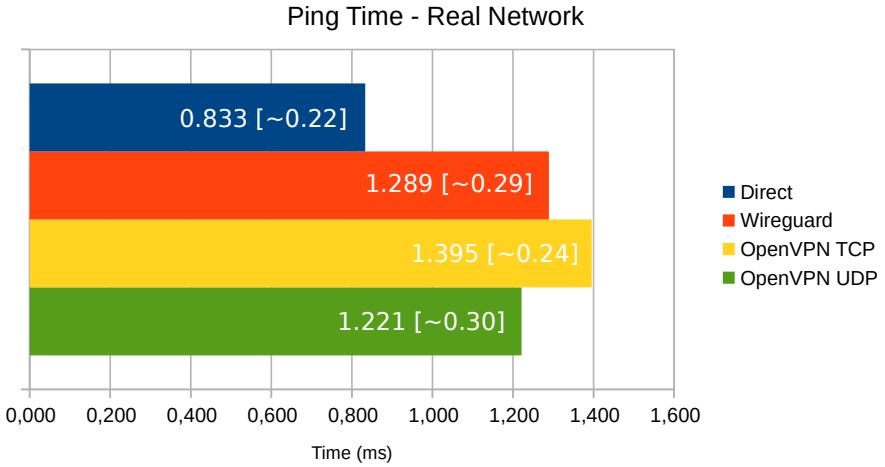

Figure 5 – Latence en réseau réel

Les résultats étiquetés *direct* représentent les performances du lien en connexion directe sans VPN.

Nous pouvons constater que les performances de débit sont relativement proches avec un léger avantage pour *WireGuard* et un retard relatif pour *OpenVPN* quand il est configuré en UDP. Les performances en temps de latence sont davantage liés aux protocoles TCP et UDP que par les VPNs eux mêmes. Ces résultats montrent que l'accélération matérielle d'AES dont dispose *OpenVPN* lui permet d'approcher les performances de *WireGuard*. Nous attirons l'attention du lecteur sur le fait que ces résultats sont fortement liés aux capacités du processeur utilisé. Ainsi, un processeur plus modeste pourra entraîner une différence de performance plus importante entre *OpenVPN* et *WireGuard*.

Afin de démontrer l'impact de l'accélération matérielle ainsi que celui du mode noyau dans le cas de *WireGuard*, nous avons effectué les mêmes tests à l'aide de la plateforme de virtualisation réseau *Nemu* [18][19], un environnement permettant d'instancier des réseaux de machines virtuelles. Nous avons ainsi connecté deux machines virtuelles *QEMU* accélérées par *KVM* tournant également sous *Debian 11* avec un noyau *Linux 5.10* équipées chacune de 2 cœurs de processeurs sans accélération matérielle des opérations cryptographiques. Les résultats en environnement virtuel sont présentés dans les figures 6 et 7.

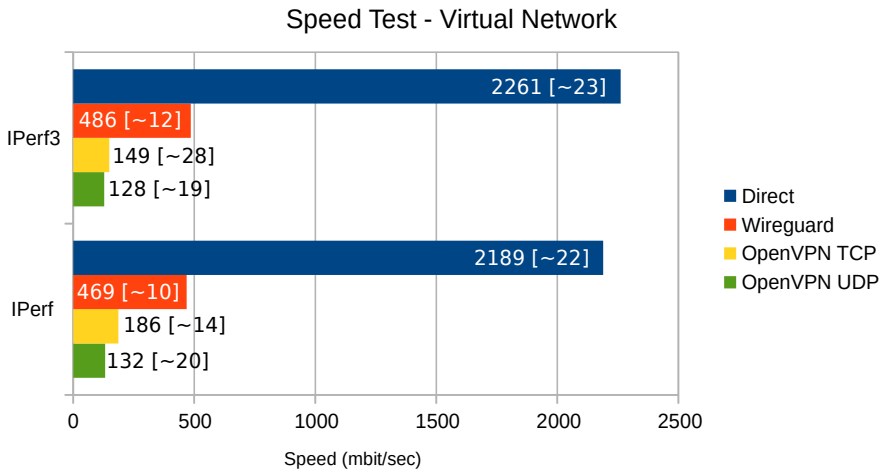

Figure 6 – Débit en réseau virtuel

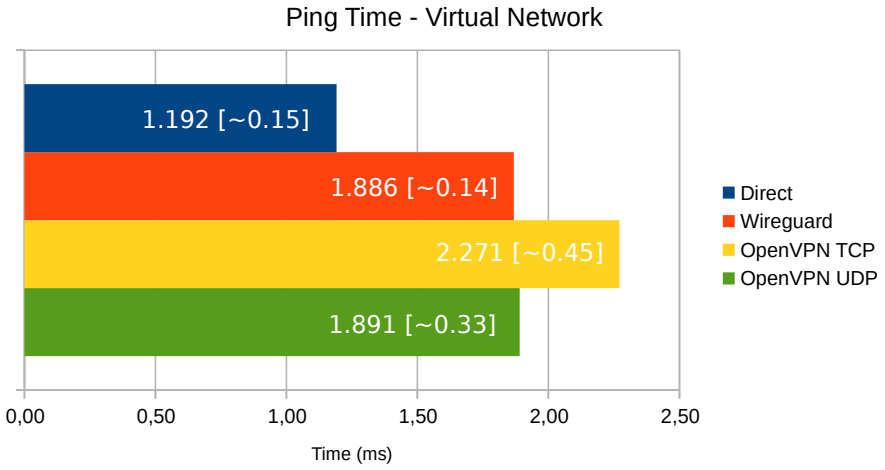

Figure 7 – Latence en réseau virtuel

Nous pouvons ici constater une différence importante des débits d'*OpenVPN* et de *WireGuard* en environnement virtuel. La principale cause est la non disponibilité de l'accélération matérielle pour les calculs cryptographiques qui oblige à effectuer ces derniers directement sur le CPU ; *OpenVPN* est ici très désavantagé. La diminution des performances de *WireGuard* en comparaison de l'expérimentation en réseau réel s'explique à la fois par l'absence accélération matérielle cryptographique mais aussi par sa sortie du mode noyau. En effet, les machines virtuelles étant exécutées en espace utilisateur, le noyau Linux tournant dans les machines virtuelles est en réalité exécuté dans un processus utilisateur de la machine hôte et non dans son propre noyau. Ainsi *WireGuard* perd l'avantage de cette exécution privilégiée dans cette expérimentation en réseau virtuel. Les résultats de latence sont globalement identiques à ceux obtenus en conditions réelles au prix d'un surcoût induit par la couche de virtualisation.

Nous pouvons ici conclure que l'accélération matérielle des calculs cryptographiques symétriques a un rôle majeur dans les performances des VPNs y compris pour *WireGuard*. Dans le cas d'une utilisation pour des machines virtuelles, *WireGuard* tire néanmoins son épingle du jeu.

4 Utilisation de WireGuard

WireGuard fonctionne sur le principe de client-serveur. Il permet de créer des interfaces virtuelles dans le noyau de manière similaire à des interfaces TUN de niveau IP. Chacune de ses interfaces virtuelles est dotée d'une paire de clés cryptographiques dédiées et représente un tunnel sur lequel peuvent être autorisés à communiquer plusieurs nœuds distants appelés *peers*. La clé publique de chacun de ses *peers* doit être déclarée explicitement dans la configuration de l'interface virtuelle. La configuration de celle-ci peut être effectuée directement en ligne de commande ou bien dans un fichier de configuration localisé le plus couramment dans /etc/wireguard.

Prenons par exemple la configuration suivante d'un serveur et de deux clients :

```
# Peer Server [real IP address: 147.210.215.26]

[Interface]
PrivateKey = OM6ee0Gj5H9t4UD8FeqXumar68I7w234D9uYn2zMIUc=
ListenPort = 51820
Address = 192.168.42.254/24
PostUp = iptables -P FORWARD ACCEPT
PostUp = echo 1 > /proc/sys/net/ipv4/ip_forward
PostDown = iptables -P FORWARD DROP

[Peer]
PublicKey = PSaOOMdLwk9SSBbjNtXA48H77RBajcVogaZ5h0G98jQ=
AllowedIPs = 192.168.42.10/32

[Peer]
PublicKey = 6f4+8iuAgZJ2hh1DxVVoHnxC7hdxuloEIv74ZXoxgCk=
AllowedIPs = 192.168.42.20/32
```

```
# Peer Client 1
[Interface]
PrivateKey = KB8ebdk5aVIZAVRRlp4o/jJF27wek68YJn95SPzFi1c=
Address = 192.168.42.10/24
[Peer]
PublicKey = tfXTAXLQTK5vBAI4F+sFDpgTokWlSdfo5uLWu06xdSE=
Endpoint = 147.210.215.26:51820
AllowedIPs = 192.168.42.0/24

# Peer Client 2
[Interface]
PrivateKey = QMUH7zcTZaQYGo+jVt+cJUV78PA4j4nBgUgscQib1Fo=
Address = 192.168.42.20/24
[Peer]
Endpoint = 147.210.215.26:51820
PublicKey = tfXTAXLQTK5vBAI4F+sFDpgTokWlSdfo5uLWu06xdSE=
AllowedIPs = 0.0.0.0/0
PostUp = ip route add default via 192.168.42.254
PostDown = ip route delete default via 192.168.42.254
```

Dans le configuration serveur ci-dessus, l'interface virtuelle est configurée sur l'IP 192.168.42.254/24 en écoute sur le port 51820. Deux *peers* clients distincts sont autorisés à communiquer avec notre interface *WireGuard*. Les communications en entrée ne sont possibles que si la correspondance entre clé publique et adresse IP source est valide avec cette configuration. Ce principe fondamental, nommé *CryptoKey Routing*, est l'association entre la clé publique d'un *peer* et les adresses IP que celui-ci est en droit d'utiliser. Le fichier de configuration d'une interface *WireGuard* permet également d'ajouter des commandes PostUp et PostDown qui seront lancées respectivement au démarrage et à l'arrêt de l'interface. La configuration des deux clients est similaire à celle du serveur à la différence que le port d'écoute est remplacé par un Endpoint dans lequel est indiquée l'IP réelle du serveur qui permet d'établir le tunnel. Le premier client n'autorise l'utilisation du tunnel *WireGuard* que pour communiquer avec le réseau 192.168.42.0/24 tandis que le second l'utilise comme passerelle par défaut, comme ce serait le cas dans l'utilisation d'une offre VPN commerciale. Les fichiers de configuration d'une interface *WireGuard* contiennent de nombreuses options dont la possibilité de rajouter une clé symétrique supplémentaire appelée PresharedKey qui devra être partagée entre clients et serveur en sus des clés asymétriques.

La commande wg permet aisément de générer une paire de clés asymétriques ainsi que la clé symétrique optionnelle :

```
> wg genkey |tee /dev/stderr |wg pubkey
gKlqFrJDnIBvSm4cnWzveUpdPuSlH8SNI+/B4Pqyw2M=   # clé asymétrique privée
YpR49+zxw+MYUt7k5eWWF0oNoWAe4BkiFm288O0woS8=   # clé asymétrique publique
> wg genpsk
NcwnLgXUanmio2T77O2Hot8DvoBZDWKt3DUHi3+Mkyg=   # clé symétrique optionnelle
```

La commande wg-quick permet quant à elle de démarrer / arrêter une interface virtuelle :

```
> ip -br addr
lo               UNKNOWN        127.0.0.1/8
eth0             UP             192.168.0.0/24
> ls /etc/wireguard
wg-jres.conf
> wg-quick up wg-jres
[#] ip link add wg-jres type wireguard
[#] wg setconf wg-jres /dev/fd/63
[#] ip -4 address add 192.168.42.0/24 dev wg-jres
[#] ip link set mtu 1420 up dev wg-jres
> ip -br addr
lo               UNKNOWN        127.0.0.1/8
eth0             UP             192.168.0.0/24
wg-jres          UP             192.168.42.0/24
> wg-quick down wg-jres
[#] ip link delete dev wg-jres
> ip -br addr
lo               UNKNOWN        127.0.0.1/8
eth0             UP             192.168.0.0/24
```

5 Exemples d'utilisation réelle

5.1 Accès administrateur distant à un centre d'enseignement supérieur

Le CREMI[1] est le centre de ressource pour l'enseignement des mathématiques et de l'informatique de l'Université de Bordeaux qui fournit des salles informatiques pour plusieurs milliers d'étudiants. Avec la généralisation du télétravail, les administrateurs systèmes et réseaux de ce centre ont mis en place un VPN *WireGuard* pour leur besoin d'administration à distance au travers d'un routeur *OPNSense* [20]. Leur configuration est à la fois disponible en IPv4 ainsi qu'en IPv6.

5.2 Durcissement d'un système d'intégration continue

WireGuard nous a été utile afin de sécuriser l'accès à notre serveur de dépôts Git ainsi que la connexion entre ce dernier et les serveurs d'intégration continue (*i.e.* CI). De cette façon, l'accès au serveur de dépôt est restreint aux seuls membres ayant accès au réseau VPN. Les performances de débits de *WireGuard* nous permettent de conserver des vitesses de transfert proches de celles du lien lors d'un *pull* ou d'un *push* mais aussi lors de la récupération des dépôts par les serveurs d'intégration continue qu'ils soient physiques ou virtuels (i.*e.* VM). Ces derniers sont également connectés à des serveurs de déploiement continu (i.e. CD) au travers d'une autre configuration *WireGuard*.

6 Conclusion

Dans cette article, nous avons présenté les fondements ainsi que le fonctionnement général de la solution libre VPN *WireGuard*. Nos expérimentations ont permis de mettre en évidence la grande importance de l'accélération matérielle des opérations de cryptographie symétrique vis à vis des performances en termes de débit. Nous avons également montré que l'implémentation de *WireGuard* ainsi que ces choix techniques permettent d'obtenir des performances de débit significativement supérieures à celles d'*OpenVPN* dans le cas d'une exécution sur machines virtuelles. Nous pouvons ainsi préconiser l'utilisation de *WireGuard* dans ce contexte afin de maximiser les vitesses de transfert.

Il serait intéressant de réaliser ces expérimentations sur un réseau *full 10 Gbit* afin de vérifier si l'accélération matérielle des opérations AES d'*OpenVPN* suffisent pour égaler celle de *WireGuard* avec Chacha20 et son exécution privilégiée en mode noyau.

1 https://www.emi.u-bordeaux.fr

7 Bibliographie

[01] Philip Christian, *VPN Usage is Increasing*, Malware Bytes Labs, https://blog.malwarebytes.com/malwarebytes-news/2021/01/vpn-usage-is-increasing-says-survey, janvier 2021

[02] Jason A. Donenfeld, *WireGuard.* https://www.wireguard.com

[03] Wikipedia, *Virtual Private Network.* https://en.wikipedia.org/wiki/Virtual_private_network

[04] Stephen Kent et Karen Seo, *Security Architecture for the Internet Protocol*, IETF RFC 4301, https://www.rfc-editor.org/rfc/rfc4301.txt, décembre 2005

[05] Xelerance, *Openswan*, https://www.openswan.org

[06] Andreas Steffen, *StrongSwan*, https://www.strongswan.org

[07] Eric Rescorla, *The Transport Layer Security (TLS) Protocol Version 1.3*, IETF RFC 8446, https://www.rfc-editor.org/rfc/rfc8446.txt, août 2018

[08] *OpenVPN Project*, https://openvpn.net

[09] Jason A. Donenfeld, *WireGuard: Next Generation Kernel Network Tunnel*, NDSS. pages 1-12 , février 2017

[10] Iperf, https://iperf.fr

[11] Trevor Perrin, *The Noise Protocol Framework*, https://noiseprotocol.org

[12] IANA, *Transport Layer Security (TLS) Parameters*, https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml, décembre 2021

[13] IANA, *Internet Key Exchange Version 2 (IKEv2) Parameters*, https://www.iana.org/assignments/ikev2-parameters/ikev2-parameters.xhtml, novembre 2021

[14] Benjamin Dowling et Kenneth G. Paterson, *A Cryptographic Analysis of the WireGuard Protocol*, Springer International Conference on Applied Cryptography and Network Security, pages 3-21, juin 2018.

[15] Benjamin Lipp, Bruno Blanchet et Karthikeyan Bhargavan, *A Mechanised Cryptographic Proof of the WireGuard Virtual Private Network Protocol,* INRIA Research Report 9269, avril 2019

[16] NIST, *Advanced Encryption Standard (AES)*, FIPS 197, novembre 2001

[17] Daniel J. Bernstein, *ChaCha, a Variant of Salsa20,* Workshop record of SASC, pages 3-5, 2008

[18] Vincent Autefage et Damien Magoni, *NEmu : un outil de virtualisation de réseaux à la demande pour l'enseignement*, JRES 10th, décembre 2013

[19] Vincent Autefage, *Network Emulator for Mobile Universes*, https://gitlab.com/v-a/nemu

[20] Franco Fichtner, Ad Schellevis et Jos Schellevis, *OPNSense*, https://opnsense.org