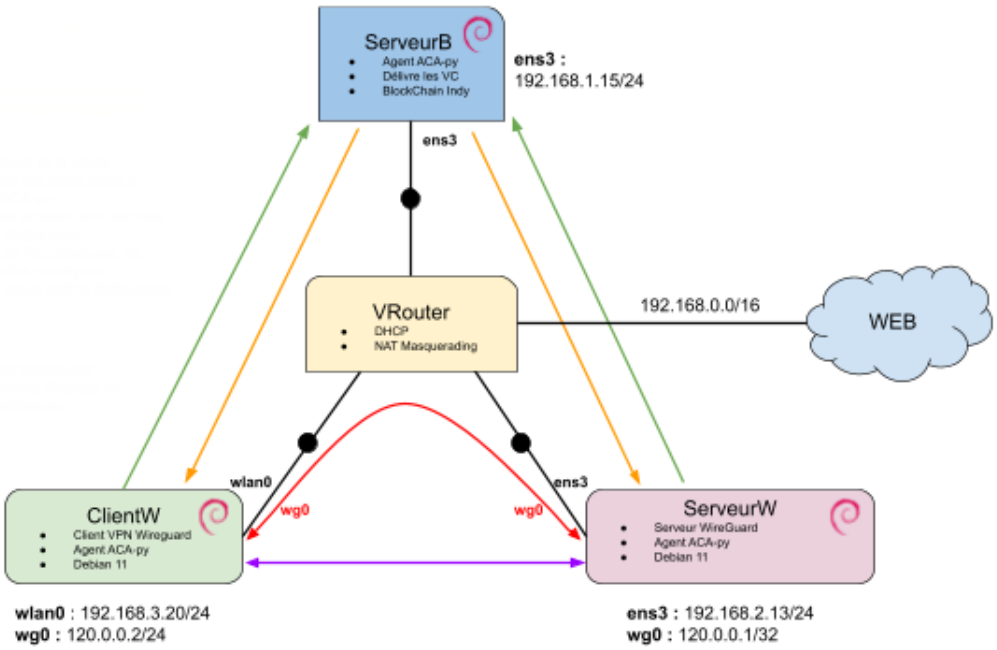
université de BORDEAUX

MASTER 2 INFORMATIQUE RCI

2021-2022

Rapport de PFE : Déploiement d'un VPN sur des équipements mobiles ou IoT



Alexis Henquinet Mohamed Diallo Sara Real Santos

Sommaire

1 Introduction

1.1 Contexte

De nos jours l'emploi de réseaux privés virtuels (VPN, Virtual Private Network) est de plus en plus démocratisé. On s'en sert généralement pour masquer son adresse ip, ou pour créer un canal sécurisé chiffré avec un destinataire. De nombreuses applications, services et protocoles de VPN différents existent, que ce soit sur Ordinateur ou Smartphone.
En tant qu'utilisateur, se connecter à un serveur VPN nécessite d'en connaitre son adresse ip ainsi qu'échanger des clés de chiffrement (symétrique ou asymétriques) avec celui-ci. Cependant pour garantir l'authenticité de la connexion, et identifier le serveur/client avec lequel le tunnel VPN s'établit, on peut recourir aux Certificats. Lorsqu'un client veut se connecter à un serveur, il lui demande son certificat afin de prouver son identité. Ce certificat étant délivré par un tier de confiance, à savoir l'Autorité de certification, le client peut donc avoir une preuve de l'identité du serveur. Cependant, ce système étant centralisé, il dépend des autorités de certification et peut présenter différents problèmes. D'une part la possible censure ou contrôle de la part de cette autorité, mais aussi le fait que si jamais cette autorité est attaquée, alors tous les certificats délivrés par celle-ci sont compromis. Pour pouvoir contrer ces difficultés, de nouvelles méthodes basée sur la décentralisation des autorités de certifications existent, comme par exemple basées sur la BlockChain.

1.2 Présentation du projet

Déploiement d'un VPN sur des équipement mobiles ou IoT est un projet dont le but est de réussir à installer et configurer le VPN WireGuard sur un client Android. Cet Android sera une machine virtuelle qui s'appuie sur les logiciels QEMU et KVM, et membre d'un réseau virtuel NEmu.
WireGuard est un VPN nécessitant des couples de clés publique/privée de chiffrement asymétrique. Afin de garantir l'authentification et l'identité du serveur VPN WireGuard auquel le client Android se connectera, cela nécessitera l'emploi de Verifiable Credentials (VC), un équivalent des Certificats mais dont l'autorité de certification repose sur la décentralisation, à savoir un noeud de blockchain déjà existant. Le projet Aries Hyperledger soutenu par la fondation Linux permet de développer des mécanismes basés sur les VCs. Nous utiliserons donc Aries Mobile Agent sur le client Android pour communiquer avec un Aries Cloud Agent relié à un réseau Hyperledger Indy dont le rôle est de délivrer les VC. Nous utiliserons également un Aries Cloud Agent sur le serveur WireGuard afin de communiquer avec le noeud Indy et récupérer un VC.

2 Étude de l'existant

2.1 Wireguard

WireGuard est un VPN fonctionnant sur la couche 3 du modèle OSI. Il est implémenté comme une interface réseau virtuelle du noyau pour Linux. Il est pensé pour remplacer les VPN IPsec et ceux basés sur TLS comme OpenVPN, tout en se voulant plus sûr, performant et facile d'utilisation de part son implémentation en moins de 4000 lignes de code facilement compréhensibles et vérifiables sur les systèmes Linux.
Il a été initialement déployé pour les systèmes Linux, mais il dispose maintenant de portages sous Android, Windows ou macOS principalement. Pour un client, WireGuard ne nécessite qu'un échange de clés publiques et d'informations de connexions comme l'adresse ip du serveur hôte. De courtes clés statiques pré-partagées en Curve25519 (basé sur Diffie-Hellman) sont utilisées pour l'authentification mutuelle. Le protocole proposé assure une confidentialité forte ainsi qu'un haut degré de dissimulation d'identité. Au niveau du transport, il utilise le cryptage authentifié ChaCha20Poly1305 pour l'encapsulation des paquets en UDP. La clé publique construite avec Curve25519 est utilisée pour créer une interface réseau qui lui est associée. Ces interfaces font partie de la 'cryptokey routing table', qui peut être configuré et à laquelle on peut ajouter des règles de routage supplémentaires. Quand des paquets sont envoyés vers une machine par un tunnel Wireguard (donc, par l'interface dans la cryptokey routing table), ils sont chiffrés avec la clé publique de la machine qui reçoit le paquet.
Il est cependant à noter que les couples de clés asymétriques publiques/privées génrérées par WireGuard à l'aide de son algorithme ne sont pas adaptables aux Certificats X509 de part leur format (32 caractères 64 bits).
Nous utiliserons pour notre projet la version mobile Android de WireGuard côté client, ainsi que la version Linux de WireGuard côté serveur.

2.2 Verifiable Credentials

Un "credential" correspond à une accréditation, un certificat ou une référence. Par exemple dans le monde physique cela peut être une pièce d'identité, un passeport, un permis ou encore un diplôme universitaire. Avec l'avènement de la BlockChain, il est maintenant possible de représenter numériquement et de manière sûre un credential. On parle alors de Verifiable Credential, car grâce à la BlockChain il est possible de les vérifier facilement et rapidement.
En Novembre 2019 le W3C a entammé une procédure de normalisation en publiant une recommandation sur les Verifiable Credentials, dans le but de garantir un format générique reconnu mondialement. Il s'agit donc d'une technologie novatrice et en pleine construction.
Il y est stipulé qu'un détenteur (ou "Holder") de VC peut générer une présentation à partager à quelqu'un voulant une preuve d'accréditation selon certaines caractéristiques, et ce sans forcément transmettre l'entièreté des caractéristiques de son VC. Par exemple pour obtenir un

service nécessitant d'être majeur, une preuve de notre âge peut nous être demandée. Il n'est alors pas obligatoire de transmettre le champ de son VC d'identité contenant sa date de naissance, mais par exemple juste prouver dans la présentation que l'on a "plus de 18 ans". Pour mieux comprendre les différents rôles et informations concernant les VCs, voici ci-dessous le schéma proposé par la W3C :

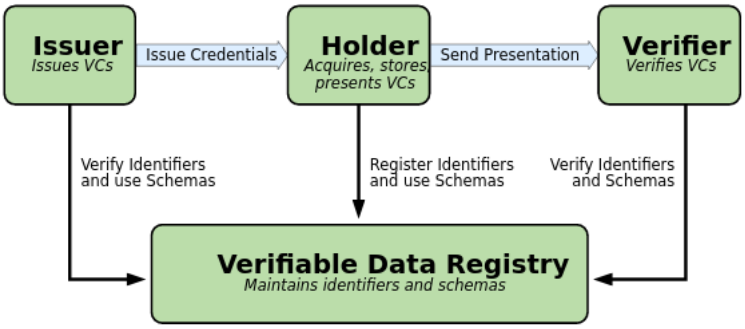


FIGURE 1 – Schema des rôles et des échanges d'information

Des mécanismes de preuves et de signatures numériques sont nécessaires afin d'assurer la protection d'un Verifiable Credential. L'obtention de la validation des preuves peut dépendre de la syntaxe de la preuve, cependant dans le cadre de ce projet les VCs correspondront à des JSON Web Tokens sécurisés par l'utilisation de JSON Web Signatures. Voici ci-dessous un exemple de de VC JWT présenté par le W3C :

```
---------- JWT header ----------
{
  "alg": "ES256",
  "typ": "JWT"
}
---------- JWT payload ----------
// NOTE: The example below uses a valid VC-JWT serialization
//       that duplicates the iss, nbf, jti, and sub fields in the
//       Verifiable Credential (vc) field.
{
  "vc": {
    "@context": [
      "https://www.w3.org/2018/credentials/v1",
      "https://www.w3.org/2018/credentials/examples/v1"
    ],
    "id": "http://example.edu/credentials/3732",
    "type": [
      "VerifiableCredential",
      "UniversityDegreeCredential"
    ],
    "issuer": "https://example.edu/issuers/565049",
    "issuanceDate": "2010-01-01T00:00:00Z",
    "credentialSubject": {
      "id": "did:example:ebfeb1f712ebc6f1c276e12ec21",
      "degree": {
        "type": "BachelorDegree",
        "name": "Bachelor of Science and Arts"
      }
    }
  },
  "iss": "https://example.edu/issuers/565049",
  "nbf": 1262304000,
  "jti": "http://example.edu/credentials/3732",
  "sub": "did:example:ebfeb1f712ebc6f1c276e12ec21"
}
---------- JWT ----------
eyJhbGciOiJFUzI1NiIsInR5cCI6IkpXVCJ9.eyJ2YyI6eyJAY29udGV4dCI6WyJodHRwczovL3
d3dy53My5vcmcvMjAxOC9jcmVkZW50aWFscy92MSIsImh0dHBzOi8vd3d3LnczLm9yZy8yMDE4L
2NyZWRlbnRpYWxzL2V4YW1wbGVzL3YxIl0sImlkIjoiaHR0cDovL2V4YW1wbGUuZWR1L2NyZWRl
bnRpYWxzLzM3MzIiLCJ0eXBlIjpbIlZlcmlmaWFibGVDcmVkZW50aWFsIiwiVW5pdmVyc2l0eUR
lZ3JlZUNyZWRlbnRpYWwiXSwiaXNzdWVyIjoiaHR0cHM6Ly9leGFtcGxlLmVkdS9pc3N1ZXJzLz
U2NTA0OSIsImlzc3VhbmNlRGF0ZSI6IjIwMTAtMDEtMDFUMDA6MDA6MDBaIiwiY3JlZGVudGlhb
FNlYmplY3QiOnsiaWQiOiJkaWQ6ZXhhbXBsZTplYjFmNzEyZWJjNmYxYzI3NmUxMmVjMjEi
LCJkZWdyZWUiOnsidHlwZSI6IkJhY2hlbG9yRGVncmVlIiwibmFtZSI6IkJhY2hlbG9yIG9mIFN
jaWVuY2UgYW5kIEFydHMifX19LCJpc3MiOiJodHRwczovL2V4YW1wbGUuZWR1L2lzc3VlcnMvNT
Y1MDQ5IiwibmJmIjoxMjYyMzA0MDAwLCJqdGkiOiJodHRwOi8vZXhhbXBsZS5lZHUvY3JlZGVud
GlhbHMvMzczMiIsInN1YiI6ImRpZDpleGFtcGxlOmViZmViMWY3MTJlYmM2ZjFjMjc2ZTEyZWMy
MSJ9.giMDNtWUgKbvWL4pteSpSkrh-lhgkhJUZ_gatHdRvEFs9_kB4G9neABvTuuQKfwERzi2KF
Qz3X0nzF-jrOO-5w
```

2.3 Aries Hyperledger

Aries Hyperledger est un projet developpé par l'Hyperledger Foundation. Hyperledger Foundation est une communauté centrée sur le développement d'outils, librairies et frameworks permettant de déployer des blockchains qui seront majoritairement utilisées par des entreprises.
Il y a différents projets lancées par la Fondation Hyperledger. Aries Hyperledger est le 13ème projet lancé par cette communauté. Aries Hyperledger est une infrastructure permettant l'échange de données en relation à une blockchain ainsi que l'échange de messages peer-to-peer.

Aries Hyperledger inclut plusieurs services dans son infrastructure :

- Une couche interface appelée **resolver** qui permet de créer et signer des transactions blockchain
- Un **wallet** sécurisé permettant de garder des secrets et autres informations
- Un système de **messagerie** encrypté pour l'échange entre clients hors blockchain
- Une implementation des **W3C Verifiable Credentials**
- Une implementation du Decentralized Key Management System (DKMS)
- Un mechanisme qui permet de construire protocoles et des API

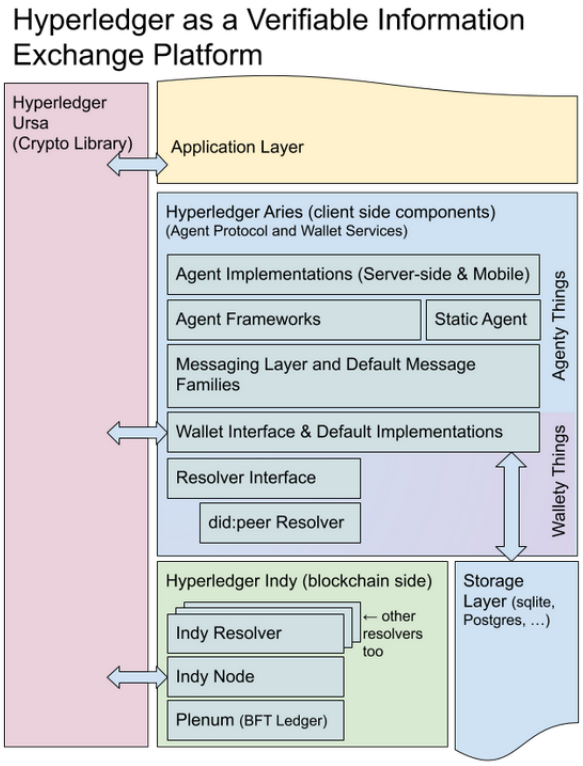


FIGURE 2 – Différents projets Aries et leur intéractions

Le plus grand objectif du projet Aries Hyperledger est de pouvoir offrir une infrastructure capable de s'adapter et de travailler avec des technologies développées par Indy ou qui se servent d'autres technologies blockchain.

2.3.1 Aries Cloud Agent

Hyperledger Aries Cloud Agent Python (ACA-Py) est une projet visé à servir de base pour contruire des Agents pouvant utiliser des Verifiable Credentials. Ses protocols et fonctionnalités de base permettent de délivrer, vérifier et stocker des Verifiable Credentials. Les agent ACA-Py peuvent contrôler des Verifiable Credentials de format Hyperledger Indy AnonCreds et de format W3C, dans ce projet nous nous intéréssons au dernier format (W3C). Aries Cloud Agent fonctionne avec des requêtes HTTP et des notifications webhook. Ceci donne la possibilité aux développeurs d'écrire un controlleur qui 'discute' avec notre agent en n'importe quel langage pouvant gérer des requêtes HTTP.
L'Agent Aries Cloud met aussi en oeuvre une interface OpenAPI REST pouvant servir à comprendre comment les protocols dans notre agent fonctionnent. Le développeur peut donc utiliser ceci pour gérer le comportement de l'agent. Voici un résume de l'architecture de l'agent Aries Cloud :

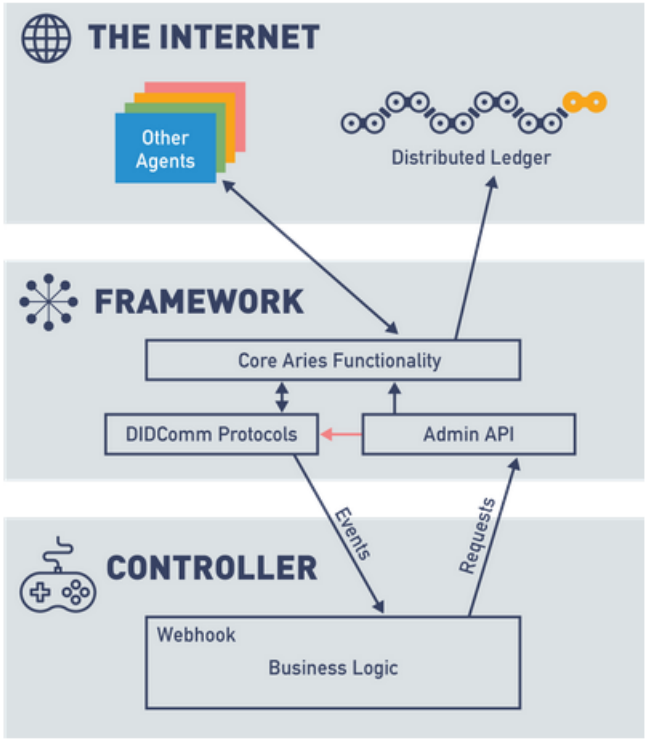


FIGURE 3 – Architecture d'un Agent Aries

Dans l'architecture, nous pouvons voir que l' Agent communique avec un Distributed Ledger, donc un Blockchain. Pour précision, dans notre projet nous utilisons Von Network comme réseau Blockchain.

Pour controler et faire des actions dans l'API il suffit juste de retrouver l'endpoint qui fera la requête que nous voulons éxecuter, le sélectionner et rentrer les champs nécessaires s'il en faut. C'est un outil très efficace pour comprendre et développer son agent.

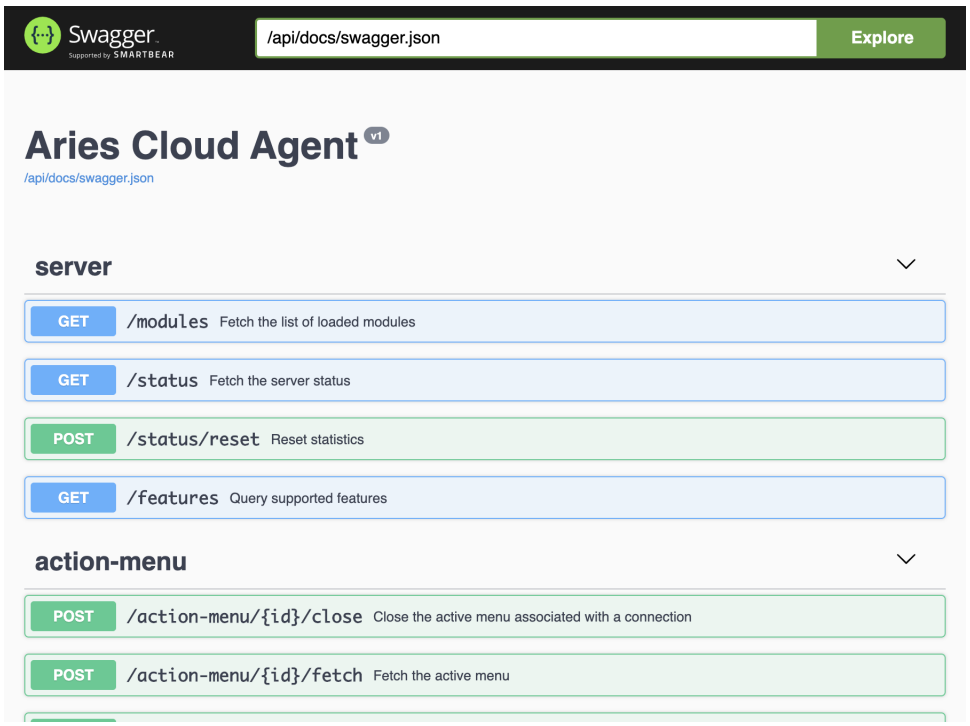


FIGURE 4 – API d'un Aries Cloud Agent

2.3.2 Aries Mobile Agent React-Native

TODO

2.4 QEMU

QEMU est un logiciel libre pouvant émuler un processeur ou une architecture différente. Il peut émuler un système ou juste le virtualiser, dépendant du système de l'hôte. QEMU peut exécuter différents systèmes d'exploitation et leurs applications de manière isolée sur une même machine physique ainsi que simuler les périphériques.
Dans notre projet nous nous sommes servis de QEMU pour émuler deux machines Debian et une machine Android. Cependant, pour des raisons techniques expliquées plus tard, nous avons dû abandonner la machine Android.

2.5 NEmu

NEmu est un environnement permettant de mettre en place des réseaux virtuels. NEmu construit une topologie d'un réseau virtuel avec des Machines Virtuelles QEMU. Il met en place aussi une API python rendant le travail plus facile. Il y a aussi la possibilité de simuler des dispositifs comme des routers ou des switchs pour administrer le réseau plus facilement. Nous avons utilisé NEmu en combinaison avec QEMU pour construire un réseau de machines virtuelles.

3 Scénarios

L'objectif de ce projet était d'installer et configurer un VPN Wireguard tout en utilisant la technologie Blockchain à la place de certificats pour s'authentifier et échanger des clés publiques.

3.1 Scénario initial

Le premier scénario de ce projet était composé de deux machines Debian, et une machine Android. Ces trois machines appartennaient toutes au même réseau. Le client Android n'ayant qu'une interface réseau, elle devait passer par un proxy sur le Serveur Wireguard pour avoir accès à Internet quand le VPN était activé.

Il y avait 3 types d'Agent à mettre en place : un Cloud Agent pour la machine Serveur Blockchain, un Cloud Agent pour la machine Serveur Wireguard et un Static Agent pour la machine Android.
Le Serveur Blockchain devait générer des clés publiques en utilisant OpenSSL, les mettre dans un Verifiable Credential et les envoyer au Client et au Serveur Wireguard.
Cependant, après des recherches, nous avons constaté que notre scénario avait des problèmes qu'ont empeché l'implémentation de celui-ci. Le principal concernait le Static Agent. Le Static Agent était sensé d'être implémenté sur le Client Android mais après quelques recherches, nous avons appris que le Static Agent n'avait pas de wallet donc, il ne pouvait pas stocker des Verifiable Credentials. Cet Agent est juste capable de recevoir des messages du Cloud Agent et en échanger avec d'autres Static Agents.

3.2 Scénario intermédiaire

Le scénario initial a dû être changé. Au lieu d'avoir un Static Agent, le Client Wireguard implemente un Mobile Agent dans notre scénario intermédiaire.
Notre scénario intermédiaire était composé de deux machines Debian implémentant toutes les deux des Cloud Agents et une machine Android implémentant un Mobile Agent.

Le projet Aries Mobile Agent est un projet qui est toujours en cours de développement. Ceci fait qu'il y a des problèmes de compatibilité entre le Mobile Agent et le Cloud Agent. Nous avons décidé de remplacer le Mobile Agent sur le Client Wireguard par un Cloud Agent. Comme le Cloud Agent ne fonctionnait pas sur Android, nous avons utilisé Debian 11 à la place.

Au niveau des clés Wireguard, OpenSSL n'est pas capable de générer des clés Wireguard. Donc, chaque machine Wireguard devra générer ses propres clés en utilisant l'algorithme de Wireguard.

3.3 Scénario fonctionnel

Notre scénario final est composé de trois machines Debian. Les trois implémentent un Cloud Agent. Chaque machine appartient à un réseau différent, et un VRouter faisant du NAT les met en relation.
Les clés Wireguard sont générées par l'algorithme de Wireguard par le Client et le Serveur Wireguard, puis envoyées au Serveur Blockchain qui les stocke dans des Verifiable Credential. Ces Verifiable Credentials sont envoyés aux machines Wireguard et utilisés pour s'authentifier et s'échanger les clés afin de mettre en place un tunnel VPN Wireguard.

4 Architecture et implémentation

4.1 Réseau virtuel NEmu

Nous avons mis en place un réseau virtuel NEmu pour mettre en relation Notre Serveur Blockchain, notre Serveur Wireguard et notre Client Wireguard. Chaque machine se trouve dans un réseau différent :

- **Serveur Blockchain** possède l'addresse IP : 192.168.1.15/24
- **Serveur Wireguard** possède l'addresse IP : 192.168.2.13/24
- **Client Wireguard** possède l'adresse IP : 192.168.3.20/24

Il faut prendre en compte aussi que notre VPN Wireguard met en place une interface réseau virtuelle du côté client et serveur Wireguard, leur adresses sont 120.0.0.2 et 120.0.0.1 respectivement.
Pour mettre tout ceci en fonctionnement, nous avons décidé d'utiliser un VRouter mis en place par NEmu. Un VRouter est un router Linux virtuel qui simplifie la gestion des réseaux virtuels. Ce router utilise DHCP pour gèrer ses tables de routage et NAT.

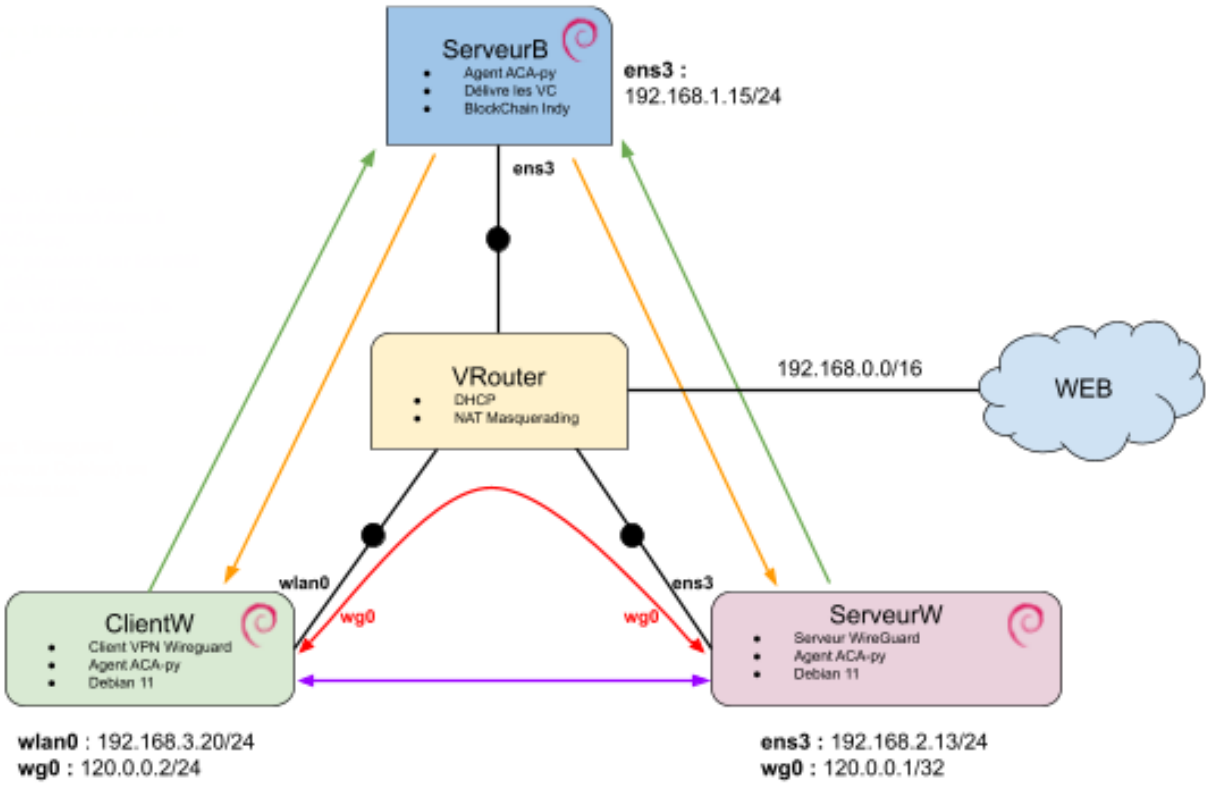


FIGURE 5 – Topologie du réseau NEmu

TODO - Expliquer le script network.py

4.2 Choix de l'implémentation

Le fonctionnement de notre implémentation est le suivant :

1. Nos Agents Client et Serveur Wireguard se connectent avec le Serveur Blockchain par Aries DIDComm
2. Le Serveur BlockChain qui est considéré le **Issuer** délivre les Verifiable Credentials au Client et Serveur Wireguard qui dans ce cas sont les **Holders**.
3. Le Serveur Wireguard et le Client établissent le canal sécurisé Aries à l'aide de l'Agent ACA-py.
   Ils peuvent ensuite prouver leur identité avec le VC qu'ils détiennent. Une fois la proof de VC effectuée, ils échangent leurs clés publiques WireGuard via le canal chiffré (DIDcomm messages).
4. Établissement du tunnel VPN avec Wireguard entre le Client et le Serveur Wireguard en utilisant les clés obtenues précédemment.

4.3 Cloud Agent du Serveur BlockChain

Le serveur Blockchain est le serveur qu'est directement relié au réseau de noeuds Blockchain. Dans le cas où on veut avoir notre propre réseau de noeuds Blockchain, le serveur Blockchain sera celui qui déploie ce réseau en utilisant Von Network pour déployer les noeuds Blockchain. Nous avons aussi la possibilité de prendre comme référence un réseau de noeuds déjà éxistant.

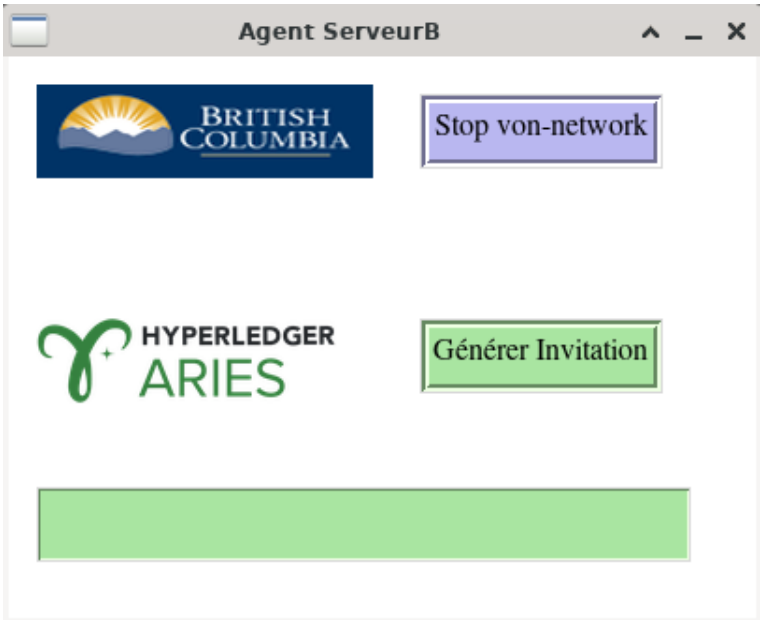


FIGURE 6 – Interface graphique de notre Serveur Blockchain

Le Serveur Blockvhain est l'acteur qu'enregistre les utilisateur auprès de la Blockchain. Il enregistre aussi les schemas et les définitions de credentials. Les Schemas et Définitions de Credentials permettent de mettre en place un format et des champs pour les Verifiable Credentials. Les Verifiable Credentials que nous avons choisi d'utiliser ont deux champs : un champ pour la **clé publique** et un autre pour le nom de la machine qui détient ce Verifiable Credential. Il y a aussi un champ contenant la signature du Verifiable Credential.

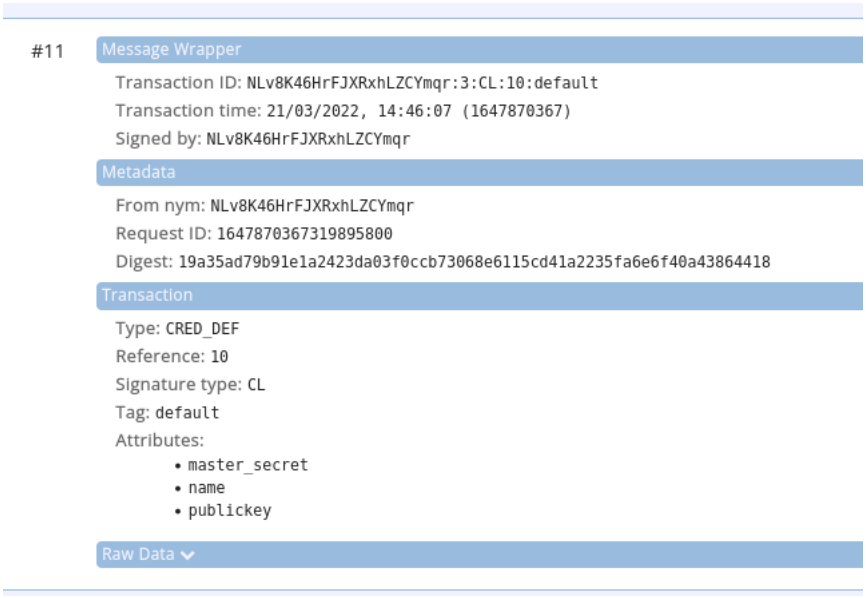


FIGURE 7 – Définition de notre Credential

4.4 Cloud Agent du Serveur WireGuard

Le Serveur Wireguard est une machine Debian qui hoste un service VPN Wireguard. Dans cette machine il y a aussi un Agent Cloud. Cet Agent Cloud est capable de communiquer avec d'autres agents, notamment dans ce cas l'Agent du Serveur Blockchain et l'Agent du Client Wireguard. Il aura de différentes intéractions avec chacun des deux agents.

Avec l'Agent Server Blockchain il aura des interactions ayant comme objectif la réception de Verifiable Credentials. Le Serveur Wireguard génère des clés Wireguard et envoie sa clé publique dans une proposal de Verifiable Credential. Un proposal est une proposition de VC basée sur une Definition de Credential (celle que nous avons enregistrée avec le Serveur Blockchain). Le Serveur Blockchain lui délivre un VC avec la clé publique et son nom.

Les échanges avec le Client Wireguard seront différents. En premier, le Serveur Wireguard doit répondre aux requêtes de preuves de la part du Client Wireguard. Le Serveur Wireguard doit en premier lieu produire une Verifiable Presentation puis l'envoyer. Cette Verifiable Presentation dépend de la requête du Client Wireguard, c'est à dire, la construction et les champs inclus dans cette Presentation dépendent de ce que le Client demande. Normalement, il va envoyer une Verifiable Presentation contenant sa clé publique et son 'nom'.
Dans la requête de preuve, le Client inclut sa clé publique Wireguard. Une fois que notre présentation est bien validée, le client aura récuperé notre clé publique. Finalement, le VPN Wireguard peut être mis en place des deux côtés.
Nous avons mis en place une interface graphique en python pour pouvoir gérer les différentes intéractions et événements.

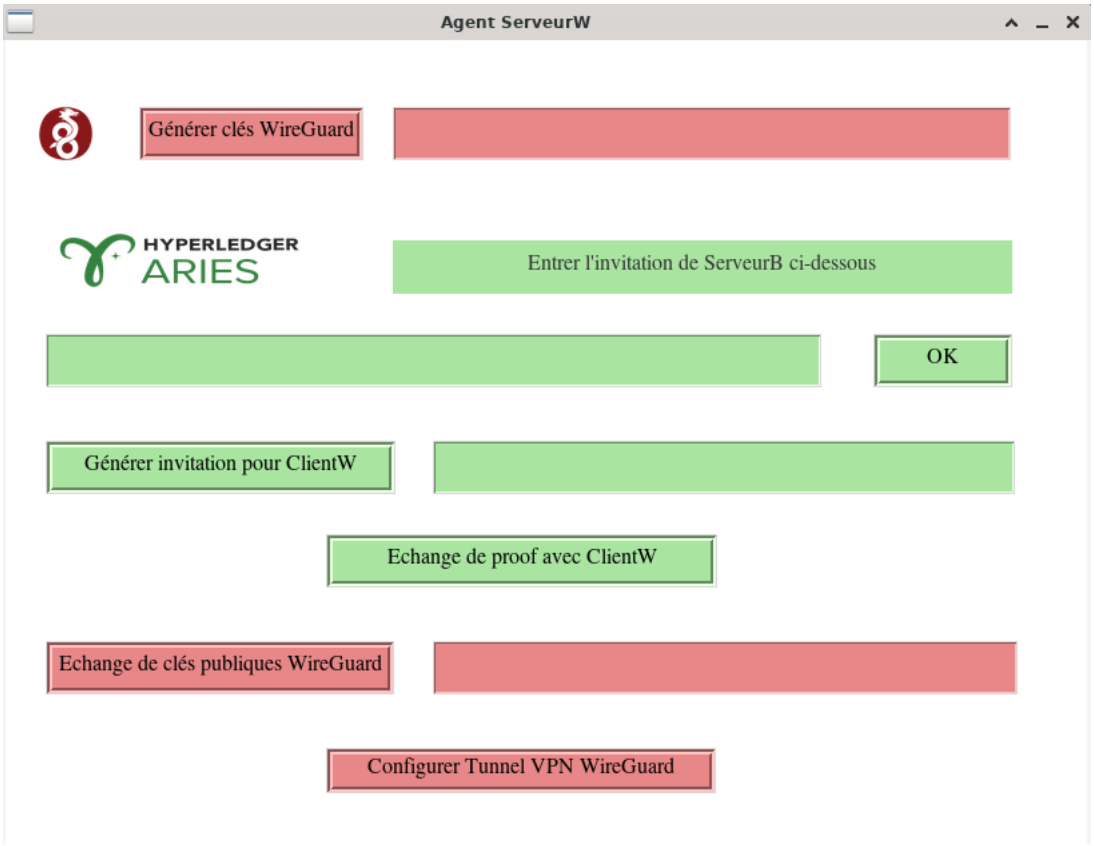


FIGURE 8 – Interface graphique de notre Serveur Wireguard

4.5 Agent du Client WireGuard

4.5.1 Mobile Agent

4.5.2 Cloud Agent

Le Client Wireguard est une machine Debian qui a comme finalité se connecter sur le VPN du serveur Wireguard.
Comme pour le Serveur Wireguard, le Serveur Blockchain doit délivrer un Verifiable Credential avec la clé publique Wireguard du client dedans. Ceci se fait avec une proposal de Verifiable Credential.
Quand le Client voudra se connecter sur le Serveur Wireguard, il devra en premier faire une requête de preuve auprès du Serveur Wireguard, contenant la clé publique du client et demandant entre autres, la clé publique du serveur. Celui-ci envoie une Verifiable Presentation avec sa clé publique Wireguard. La requête du client contennait aussi sa clé publique, donc quand la Presentation est validé, le serveur et client Wireguard ont tous les deux les clés de l'autre.

5 Analyse du fonctionnement & Tests

TODO

6 Conclusion

TODO

6.1 Limitations

TODO

6.2 Extensions

TODO

7 Bibliographie

— WireGuard : https://www.wireguard.com/ (consulté le 06/03/2022)
— Verifiable Credentials : https://www.w3.org/TR/vc-data-model/ (consulté le 06/03/2022)
— Aries Cloud Agent : https://github.com/hyperledger/aries-cloudagent-python(consultéle15
03/2022
— QEMU : https://www.qemu.org/ , https://wiki.qemu.org/Main_Page
— NEmu : https://gitlab.com/v-a/nemu

8 Annexe

TODO