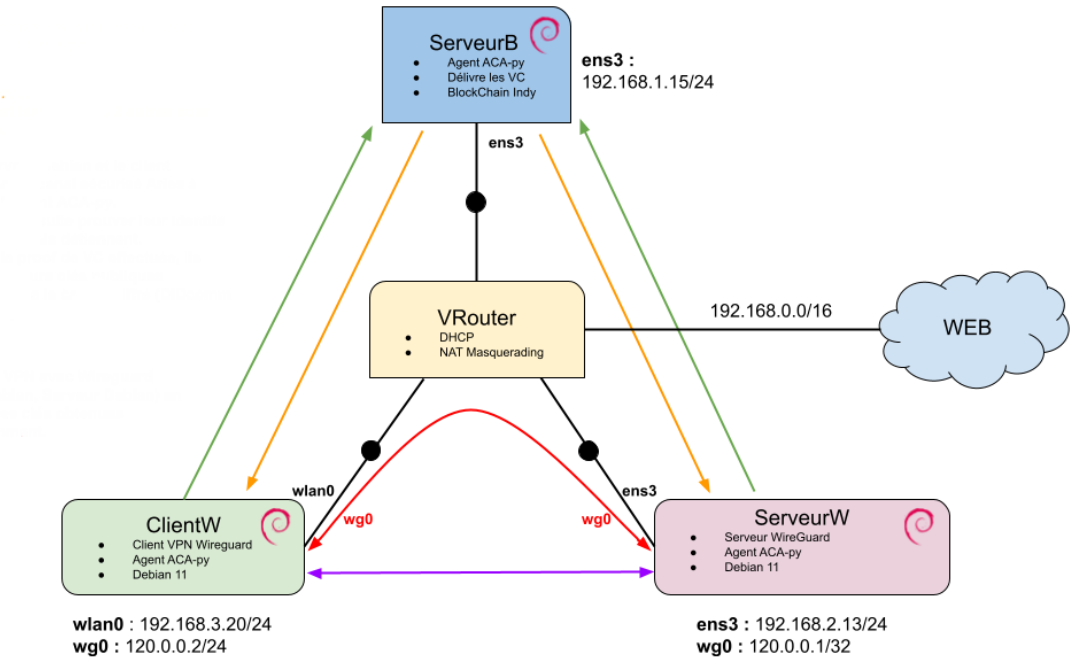
université de BORDEAUX

PROJET DE FIN D'ÉTUDE - MASTER 2 RCI

Sujet proposé par : Damien Magoni & Vincent Autefage

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 (symétriques 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 tiers 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 basées sur la décentralisation des autorités de certifications existent, comme par exemple celles basées sur la BlockChain.

1.2 Présentation du projet

Déploiement d'un VPN sur des équipements 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 configurée 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 :

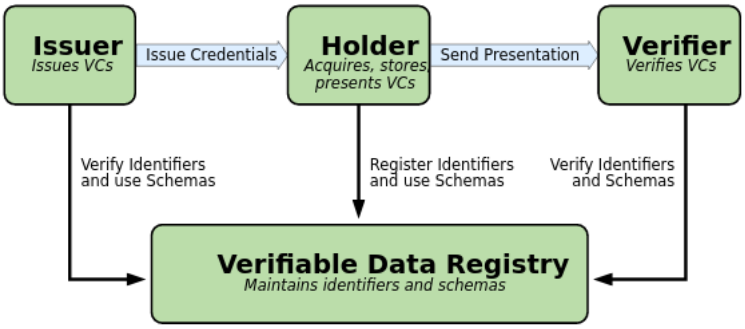


FIGURE 1 – Schema des rôles et des échanges d'information

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 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
FNlYmplY3QiOnsiaWQiOiJkaWQ6ZXhhbXBsZTplYmZlYjFmNzEyZWJjNmYxYzI3NmUxMmVjMjEi
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 lancés par la Fondation Hyperledger. Aries Hyperledger est le 13ème projet fondé par cette communauté. Aries Hyperledger est une infrastructure permettant l'échange de données en relation à une blockchain ainsi que l'échange de messages en 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** chiffrée pour l'échange entre clients hors blockchain.
- Une implémentation des **W3C Verifiable Credentials**
- Une implémentation du Decentralized Key Management System (DKMS).
- Un méchanisme qui permet de construire des protocoles et des API.

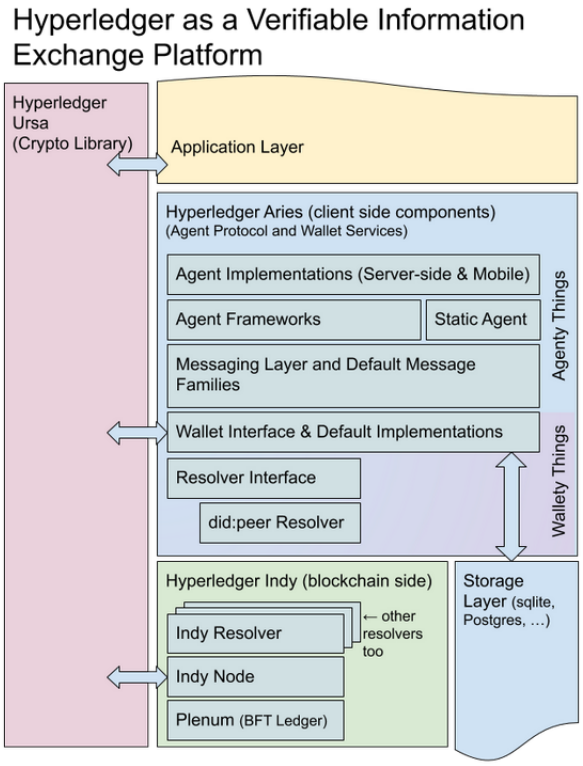


FIGURE 2 – Différents projets Aries et leurs intéractions

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 un projet visant à servir de base pour contruire des Agents pouvant utiliser des Verifiable Credentials. Ses protocoles 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 intéressons au dernier format proposé, le W3C.
Aries Cloud Agent fonctionne avec des requêtes HTTP et des notifications webhook. Ceci donne la possibilité aux développeurs d'écrire un contrôleur qui dialogue avec notre agent en n'importe quel langage permettant de gérer des requêtes HTTP.
L'Agent Aries Cloud met aussi en oeuvre une interface OpenAPI REST pouvant servir à comprendre comment les protocoles dans notre agent fonctionnent. Le développeur peut donc utiliser ceci pour gérer le comportement de l'agent. Voici un résumé de l'architecture de l'agent Aries Cloud :

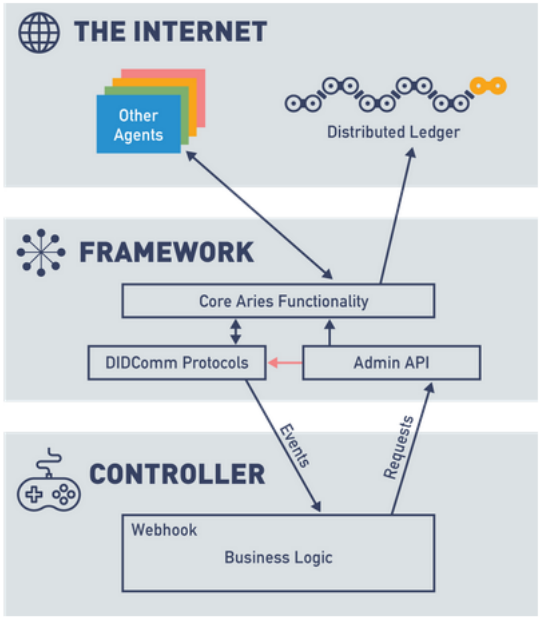


FIGURE 3 – Architecture d'un Agent Aries

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

Pour contrôler et agir dans l'API il suffit juste de retrouver l'endpoint qui se charge de la requête que nous voulons exécuter, le sélectionner et rentrer les champs nécessaires s'il en faut. C'est un outil très efficace pour comprendre et développer son propre agent. On peut en voir un exemple ci-dessous :

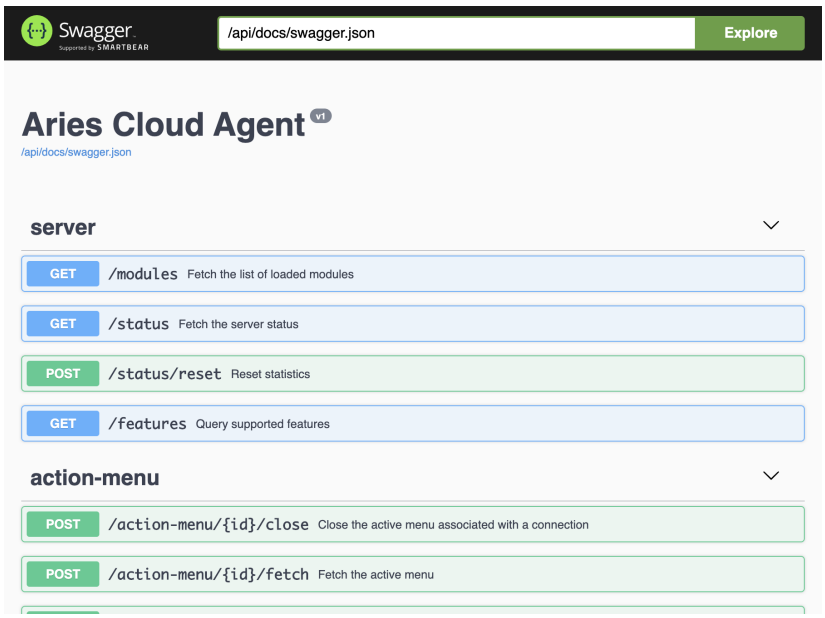


FIGURE 4 – API d'un Aries Cloud Agent

2.3.2 Aries Mobile Agent React-Native

L'Aries Mobile Agent (ou Aries Bifold) est une application Open Source développée sur React Native 0.64.1 qui a pour but de regrouper les efforts de la communauté Aries Hyperledger orientée application mobile vers un projet centralisé, afin d'éviter la duplication du code et les similitudes entre les projets. Ce projet est aussi destiné à aider d'autres projets spécifiques voulant utiliser sur mobile les technologies de la fondation Aries Hyperledger sans avoir à redéfinir la complexité interne des agents Aries. Il est conçu pour permettre de démarrer rapidement un projet en ayant un agent Aries de base.

Aries Mobile Agent est basé sur deux dépendances principales qui sont les suivantes :

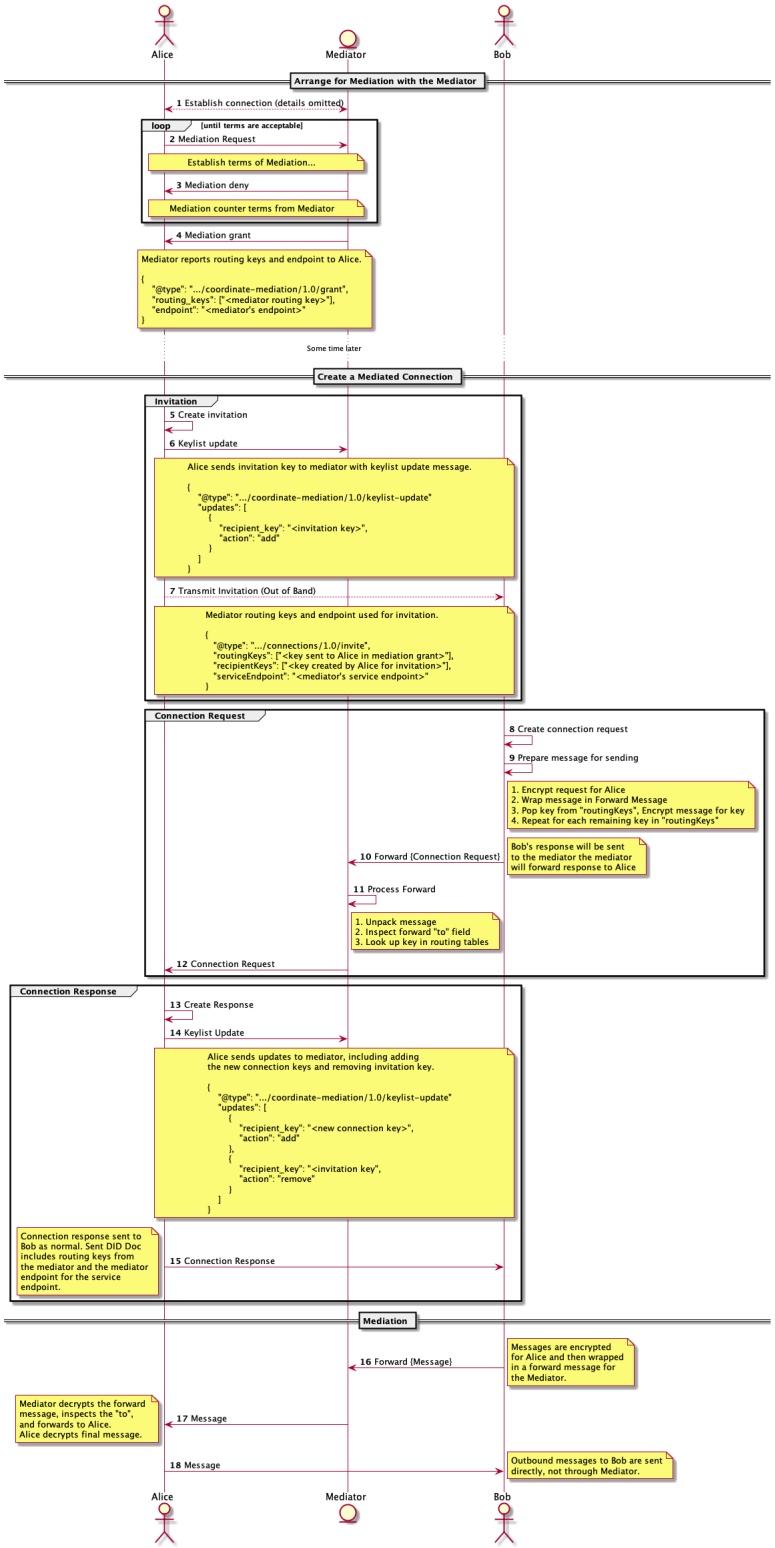
— Aries Framework Javascript, un framework écrit en TypeScript pour la création d'agents Self Sovereign Identity. SSI est une approche visant l'idée que chacun contrôle ses propres données informatiques, ainsi qu'un contrôle de quand et comment elles sont fournies aux autres. De plus, lorsqu'une donnée est partagée, cela doit être fait avec fiabilité. Avec SSI, il n'y a pas d'autorité centrale détenant les données les transmettant à d'autres entités sur demande. Vous seul en êtes le possesseur. Ce framework a également pour but d'utiliser les services DIDComm qui visent à être en conformité avec les normes définies dans les RFC d'Aries.

— Indy-sdk-react-Native : Il s'agit d'un Wrapper de Indy destiné à React Native. Son rôle est d'implémenter les fonctionnalités de Indy SDK afin qu'elles soient utilisables avec React Native.

  React Native est un framework d'applications mobiles Open Source conçu par Facebook. Il est utilisé pour développer des applications natives pour Android et iOS. Aries Bifold étant développé en React native 0.64.1, il cible l'API 29.0.3 d'Android, tandis que sur IOS il vise la version 10.0+ et il ne peut être utilisé que sur des appareils physiques pour le moment. La virtualisation est possible que sur Androidx86.

  Aries Bifold nécessite un médiateur pour fonctionner. Le mobile Agent est configuré pour utiliser la médiation implicite. Il utilise par défaut Indicio Public Mediator, médiateur public de tests en ligne basé sur un Agent ACA-Py. Un médiateur est un

agent dont le but est de faire l'intermédiaire entre un Mobile Agent et un autre Agent (par exemple un Cloud Agent ACA-py), en relayant les messages transmis entre eux. Une connexion DIDComm est établie avec le médiateur. Ci-dessous nous pouvons voir le principe et fonctionnement d'un médiateur entre deux Agents Alice et Bob comme cela est présenté dans la norme "Aries RFC 0211 : Coordinate Mediation Protocol" [1] :



1. https://github.com/hyperledger/aries-rfcs/blob/main/features/0211-route-coordination/README.md

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 Debian11 et une machine Androidx86. Cependant, pour des raisons techniques expliquées plus tard, nous avons dû remplacer la machine Androidx86 par une autre machine Debian11.

2.5 NEmu

NEmu pour Network Emulator for Mobile Universes est un environnement permettant de mettre en place des réseaux virtuels développé par Monsieur Vincent Autefage. NEmu permet de construire un environnement virtuel distribué ne nécessitant pas de droits d'administration pour fonctionner. Il contrôle un ensemble de machines virtuelles QEMU dans le but de construire une topologie de réseau virtuel. Il dispose d'une API python rendant le travail plus facile. Il dispose également de fonctionnalités de simulation de dispositifs réseaux comme des routeurs, des switchs ou encore des Smartphones pour étendre ses fonctionnalités et administrer avec plus de facilité un réseau.
Dans le cadre de ce projet nous devons utiliser NEmu pour construire le réseau virtuel sur lequel nous simulerons les interactions entre Agents Aries et le VPN WireGuard.

3 Conduite de projet & 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. Au niveau de WireGuard, il s'agit d'un VPN entièrement fonctionnel, disponible sur plusieurs plateformes et disposant d'une base de documentation importante. La difficulté de ce projet résidait surtout dans la partie Aries Hyperledger et BlockChain, des concepts nouveaux et encore en développement. Ce projet devait à l'origine être en deux parties, avec d'un côté le réseau virtuel composé d'Agents Aries et de VPN WireGuard (celui-ci), et d'un autre un lien avec un autre projet se concentrant sur le noeud de BlockChain Indy. La partie BlockChain n'ayant pas trouvée preneur, c'est en collaboration avec nos client à l'origine de ce projet, Messieurs Damien MAGONI et Vincent AUTEFAGE, qu'elle serait développée. Des réunions hebdomadaires en visio-conférence ont donc eu lieu tout au long du projet avec nos clients, initialement pour se mettre en accord sur les intéractions entre les deux projets.
Ce projet a vu son axe principal, les technologies utilisées et donc les User Stories être modifiées plusieurs fois, en accord avec nos clients. Pour mener à bien le projet, nous avons choisi

d'adopter une méthodologie agile suivant un processus itératif hebdomadaire. Les Users Stories (ou Scénarios) découpées en tâches ont été gérées depuis un Kanban permettant d'en voir la progression. L'objectif était que chaque semaine nous puissions réaliser une démo ainsi qu'un résumé des tâches accomplies et celles à faire pour les suivantes en accord avec nos clients. Nous avions à l'origine prévu un Gantt prévisionnel discuté en équipe que l'on peut retrouver ci-dessous :

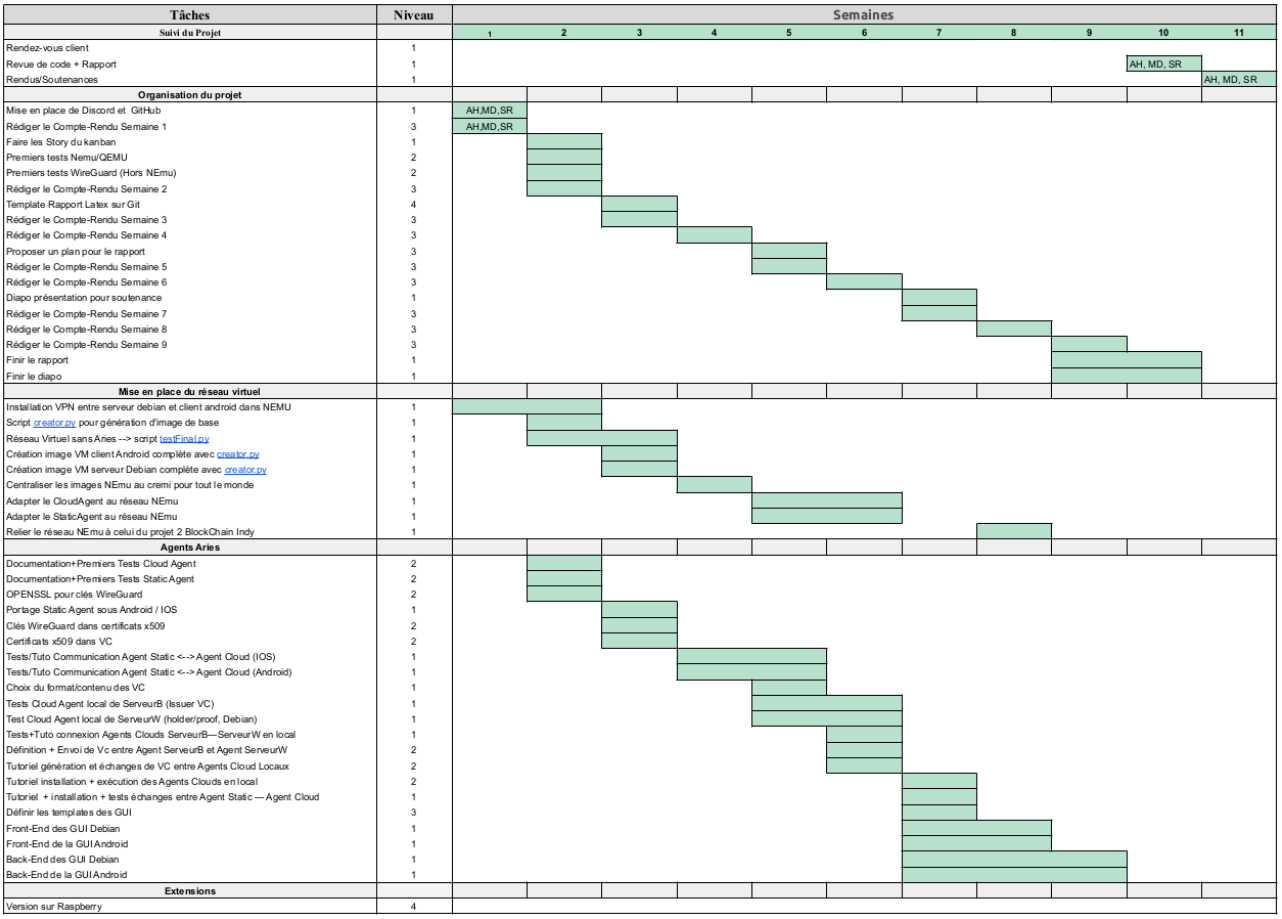


FIGURE 5 – Gantt prévisionnel du projet

On constate sur ce Gantt que nous avions prévu à l'initiale de découper les tâches du projet en 3 groupes qui sont **Organisation du projet**, **Mise en place du réseau virtuel** et **Agents Aries**. Cela nous a permis de nous séparer le travail en fonction de ce que chacun préfère réaliser, que ce soit plutôt la partie réseau virtuel NEmu ou la partie développement des Agents Aries. Nous avons aussi défini un Niveau de priorité/importance de la tâche à réaliser, allant de 1 pour le plus important à 4 pour le moins important. Ces types de tâches ainsi que le niveau de priorité se retrouvent également sur le Kanban en suivant des codes couleurs.

3.1 Scénario initial

Le premier scénario de ce projet était composé de deux machines Debian11 et d'une machine Androidx86. Ces trois machines appartenaient toutes au même réseau. Afin de fournir à la fois un accès à internet et au réseau local à l'android ne disposant que d'une interface réseau, le serveur Debian se chargeait de faire serveur proxy.

Il y avait 3 types d'Agent à mettre en place : un Cloud Agent pour la machine Serveur Blockchain relié au projet 2, un Cloud Agent pour la machine Serveur Wireguard et un Static Agent pour la machine Android. Le Serveur Blockchain devait générer des clés publiques compatibles avec WireGuard en utilisant OpenSSL, les mettre dans un Verifiable Credential et les envoyer au Client et au Serveur Wireguard. Ces deux derniers devaient alors extraire les clés publiques des VC puis configurer le tunnel VPN. On peut voir ci-dessous le plan de la topologie réseau qui avait été pensé pour correspondre à ce scénario :

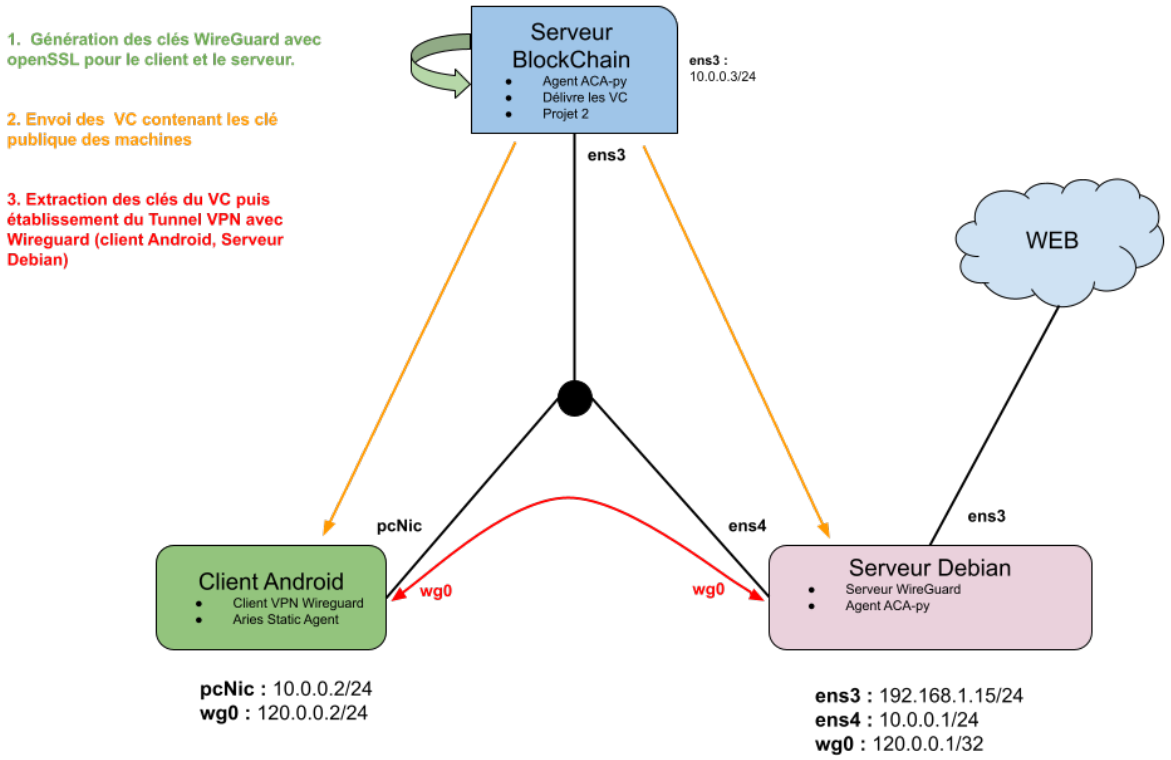


FIGURE 6 – Schéma du scénario initial

Cependant, après des recherches nous avons constaté que ce scénario avait des problèmes limitant son implémentation. Le principal problème concernait le Static Agent. Le Static Agent était sensé être implémenté sur le Client Android, mais nous avons découvert après s'être penchés en détail sur sa documentation ainsi qu'effectué des tests qu'il n'avait pas de wallet. Il ne peut donc pas stocker des Verifiable Credentials. Cet Agent a été pensé seulement pour établir une communication DIDComm avec un CloudAgent, et échanger en peer-to-peer de simples messages DIDComm.

3.2 Scénario intermédiaire

Après avoir testé et démontré que le Static Agent n'était pas utilisable dans le cadre de ce projet, nous avons du modifier notre scénario initial. Le Static Agent sur le client Mobile

Android a été remplacé par un projet novateur cherchant à implémenter un équivalent de Cloud Agent sur Mobile, à savoir le projet Aries Mobile Agent React Native.
Notre scénario intermédiaire était donc composé de deux machines Debian11 implémentant toutes les deux des Cloud Agents, et une machine Androidx86 implémentant un Mobile Agent. Au niveau de l'accès à internet et de la configuration des interfaces/adresses ip, un VRouter a été ajouté pour gérer les services dhcp, dnsmak et nat masquerading. On peut voir ci-dessous le schéma prévu pour ce scénario :

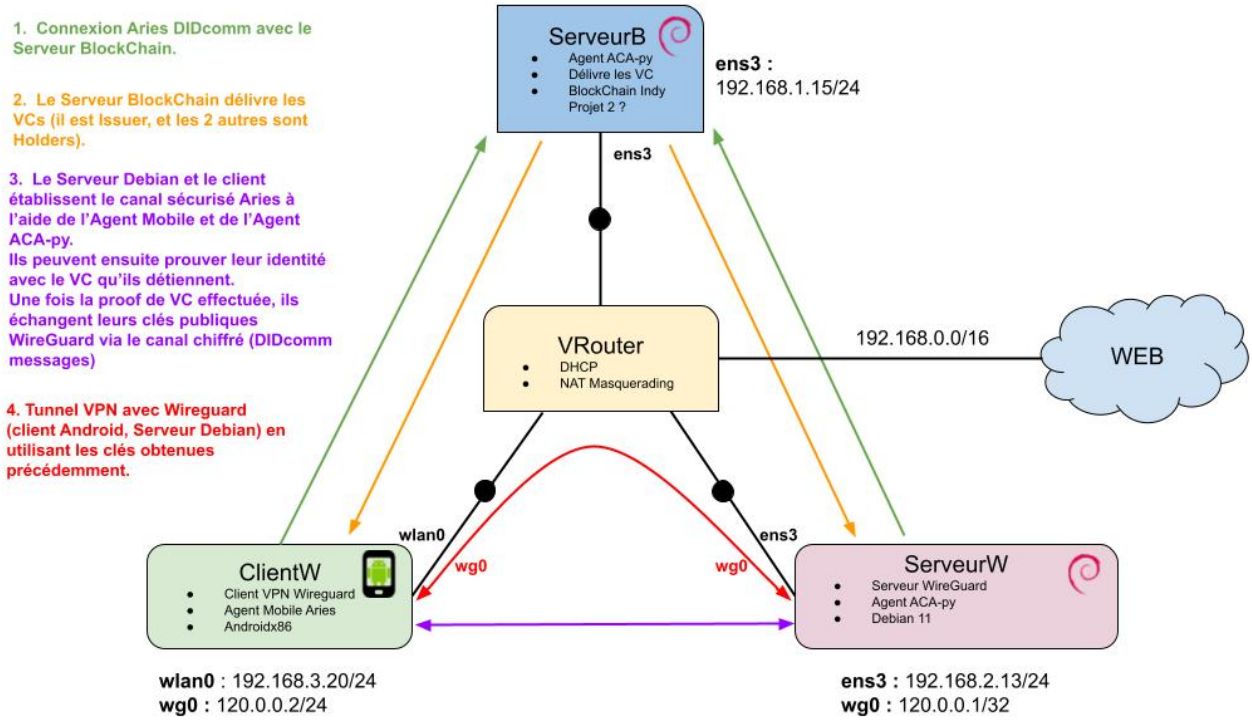


FIGURE 7 – Schéma du scénario intermédiaire

Malheureusement lors des tests et de l'implémentation du Mobile Agent dans le but de l'adapter à notre projet, nous nous sommes heurtés à de nombreux problèmes. Le projet Aries Mobile Agent est un projet réçent toujours en cours de développement dont le but est de centraliser toutes les avancées dans les portage sous mobile d'Agents Aries Hyperledgers. Comme nous le détaillerons plus tard dans la partie lui étant concernée, nous nous sommes heurtés à des problèmes de compatibilité entre ce Mobile Agent et les Cloud Agents de nos machines. Nous avons donc décidé de remplacer ce Mobile Agent sur le Client Wireguard par un Cloud Agent classique. Comme le Cloud Agent n'est pas prévu pour fonctionner sur Android, nous avons utilisé Debian11 à la place.

Au niveau des clés Wireguard, OpenSSL n'est pas capable de générer des clés Wireguard. En effet, WireGuard utilise un Algorithme propriétaire pour générer des paires de clés X25519 qui sont une méthode d'accord de clé et donc pas utilisables pour la signature. 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 Debian11. Les trois implémentent un Cloud Agent. Chaque machine appartient à un sous-réseau différent du VRouter possédant les services dnsmask, dhcp et NAT Masquerading. Le routeur garantit l'accessibilité entre les différentes machines virtuelles ServeurB, ServeurW et ClientW.
Les clés Wireguard sont générées en utilisant l'algorithme de Wireguard par le ClientW et le ServeurW, 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. Dans ce scénario, il a été convenu que les deux machines WireGuard doivent s'authentifier mutuellement.

4 Architecture et implémentation

4.1 Réseau virtuel NEmu

Nous avons mis en place un réseau virtuel à l'aide de NEmu pour mettre en relation notre Serveur Blockchain ServeurB, notre Serveur Wireguard ServeurW et notre Client Wireguard ClientW. Chaque machine se trouve dans un sous-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 entre le client et le serveur, leurs adresses virtuelles sont 120.0.0.2 et 120.0.0.1 respectivement.
Afin de faciliter la configuration des adresses et l'accès à internet, nous avons décidé d'utiliser un VRouter de NEmu. Un VRouter est un router Linux TinyCore virtuel qui simplifie la gestion des réseaux virtuels à l'aide de ses services dnsmask, dhcp et nat masquerading.

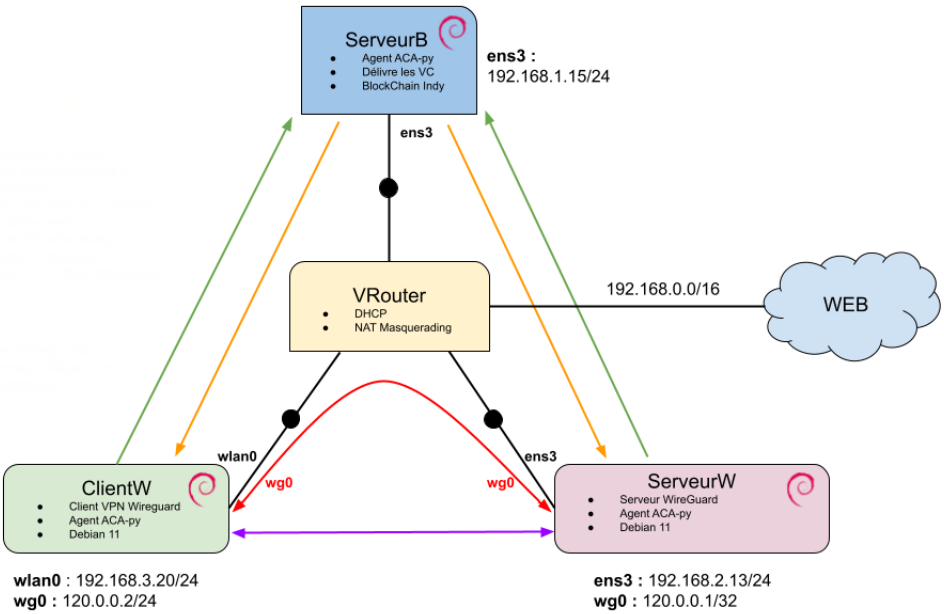


FIGURE 8 – Topologie du réseau NEmu

Le script de configuration du réseau virtuel network.py a été défini pour être utilisable au CREMI en se basant sur une seule image debian11.img (et android.img à l'origine) de base pour toutes les VMs dans /net/stockage/PFE-VPN-2022/. Cela nous as permis d'éviter d'avoir à nous transférer à chaque fois en local sur nos machines la nouvelle image de base. Par défaut le script est configuré avec un VHostConf attribuant 2 coeurs de CPU à chaque VM, ce qui suffit largement pour notre projet.
Afin de générer les images de base debian11.img et android.img nous avons réalisé un script creator.py.

4.2 Choix de l'implémentation

Au niveau des technologies utilisées pour chaque machine nous retrouvons :

- **ServeurB :** Un von-network dockerisé permettant d'obtenir un noeud de blockchain Indy + Un Aries Cloud Agent Python.
- **ServeurW :** Un Aries Cloud Agent Python + un Serveur WireGuard.
- **ClientW :** Un Aries cloud Agent Python + un Client WireGuard.

Le fonctionnement de notre implémentation est le suivant :

1. Nos Agents Client et Serveur Wireguard génèrent leurs paires de clés WireGuard, se connectent en Aries DIDComm avec le Serveur Blockchain à l'aide de l'invitation qu'il a généré, puis ils émettent une demande de VC en donnant leur clé publique WireGuard pour qu'elle y soit intégrée.

2. Le Serveur BlockChain qui est considéré comme **Issuer** délivre les Verifiable Credentials au Client et Serveur Wireguard selon le modèle demandé en y intégrant les clés publiques. ClientW et ServeurW sont donc des **Holders** de VC.
3. Le Serveur Wireguard et le Client établissent le canal sécurisé Aries à l'aide de l'Agent ACA-py. Cela nécessite que ServeurW génère une invitation à ClientW.
   Ils peuvent ensuite prouver leur identité avec le VC qu'ils détiennent en faisant deux proofs exchanges réciproques. De ce proof exchanges ils peuvent extraire la clé publique WireGuard contenue dans le VC de leur hôte.
4. Établissement du tunnel VPN avec Wireguard entre le Client et le Serveur Wireguard en utilisant les clés obtenues précédemment.

Nous pouvons voir ci-dessous le diagramme de Séquence général des interactions entre les différents agents que comporte notre projet :

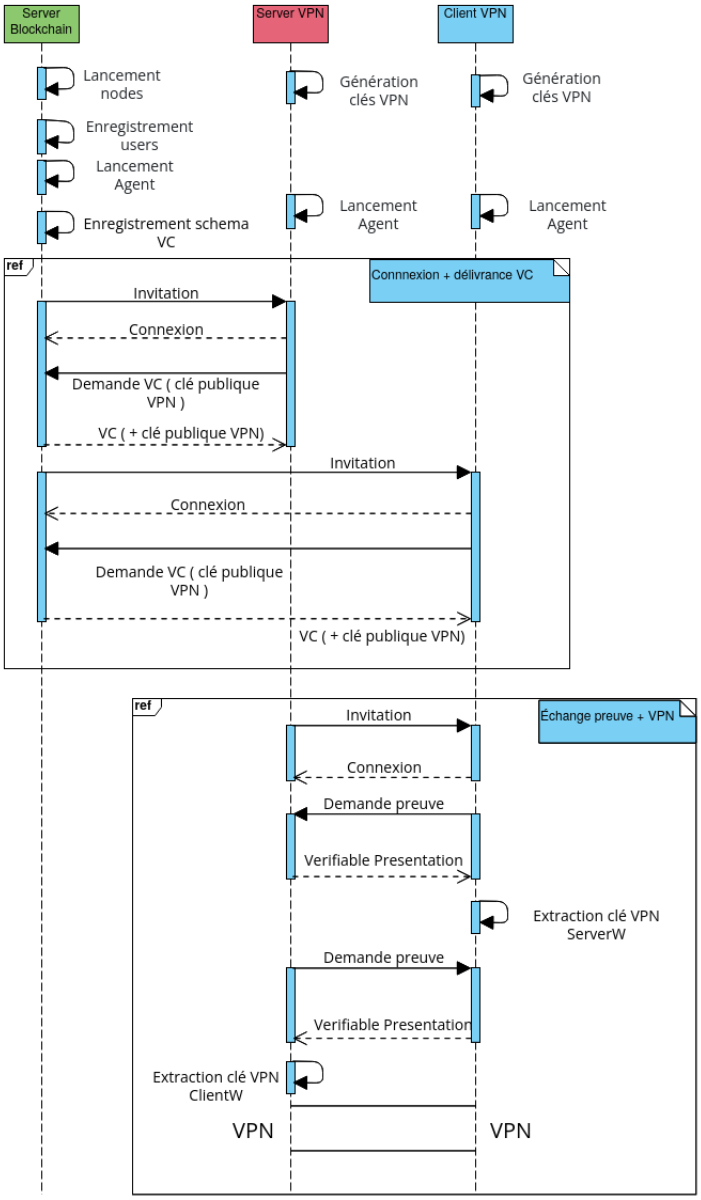


FIGURE 9 – Diagramme de séquence général

4.3 Cloud Agent du Serveur BlockChain

Le serveur Blockchain est le serveur étant directement relié au réseau de noeuds BlockChain Indy. Dans le cadre de ce projet nous voulions faire tourner notre propre réseau BlockChain sur ServeurB, c'est pourquoi nous avons utilisé le projet VON-Network [2] dockerisé. Nous avions aussi la possibilité de prendre comme référence un réseau de noeuds déjà existant hébergé par une autre machine en renseignant son URL, comme par exemple celui de British Columbia Digital Government [3].
Le Serveur BlockChain ou ServeurB est celui qui enregistre tous les utilisateurs auprès de la blockchain, ainsi que le schéma et la définition du credential utilisé pour notre projet afin d'obtenir un VC contenant une clé publique WireGuard. Il nécessite donc d'avoir un Agent ACA-py ServeurB.
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.

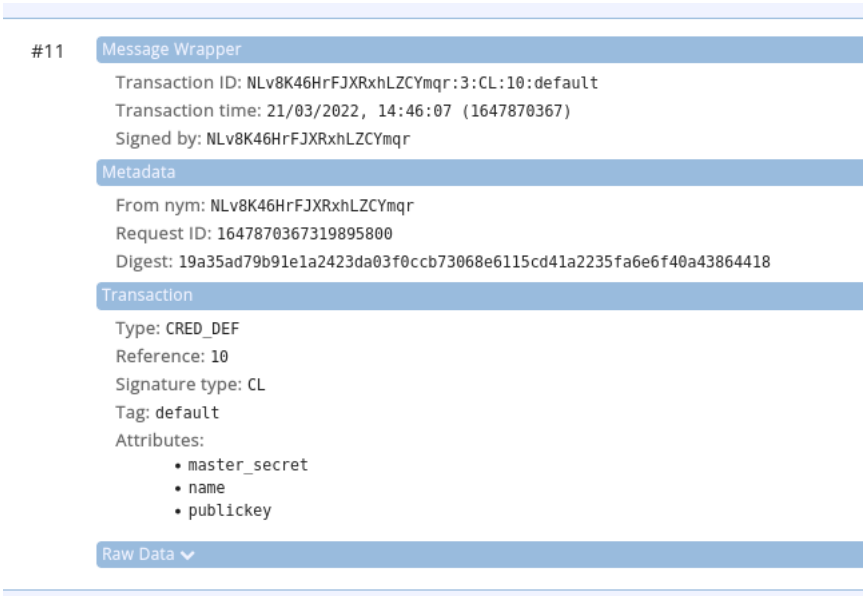


FIGURE 10 – Définition de notre Credential

ServeurB dispose également de sa propre application possédant une petite interface graphique qui simplifie les actions à effectuer qu'on peut voir ci-dessous :

Cette interface permet de stopper le von-network dockerisé local si on le souhaite car il s'agit d'un processus indépendant lancé au démarrage de l'application. Nous avons ajouté cette option pour des raisons pratiques lors du développement de l'agent. On peut aussi y voir un bouton pour générer des invitations à l'attention de ClientW et ServeurW. Ces invitations s'afficheront dans la zone de texte (vide sur cette image), mais seront également enregistrées dans **ressources/invitation.json**.

On retrouve ci-dessous le diagramme d'états de ServeurB :

2. VON-Network : `https://github.com/bcgov/von-network`
3. GreenLight Ledger : `http://greenlight.bcovrin.vonx.io/`

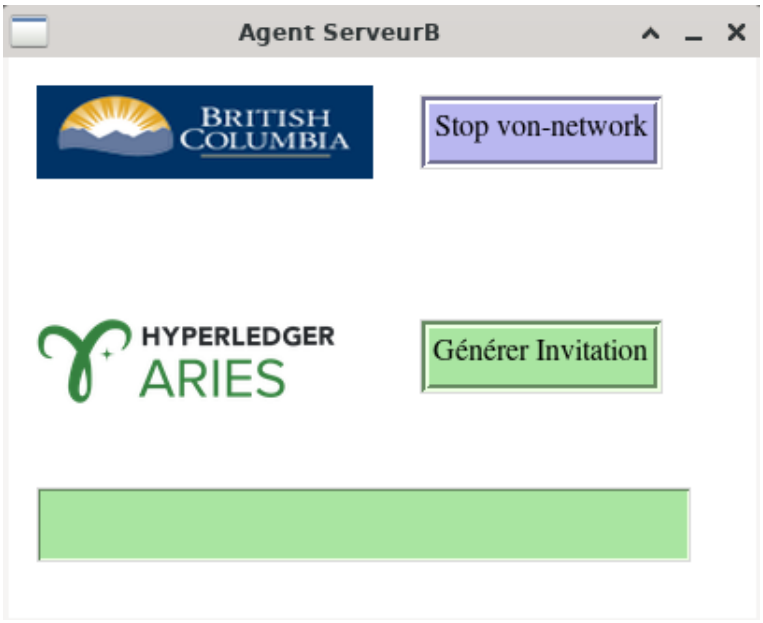


FIGURE 11 – Interface graphique de notre Serveur Blockchain

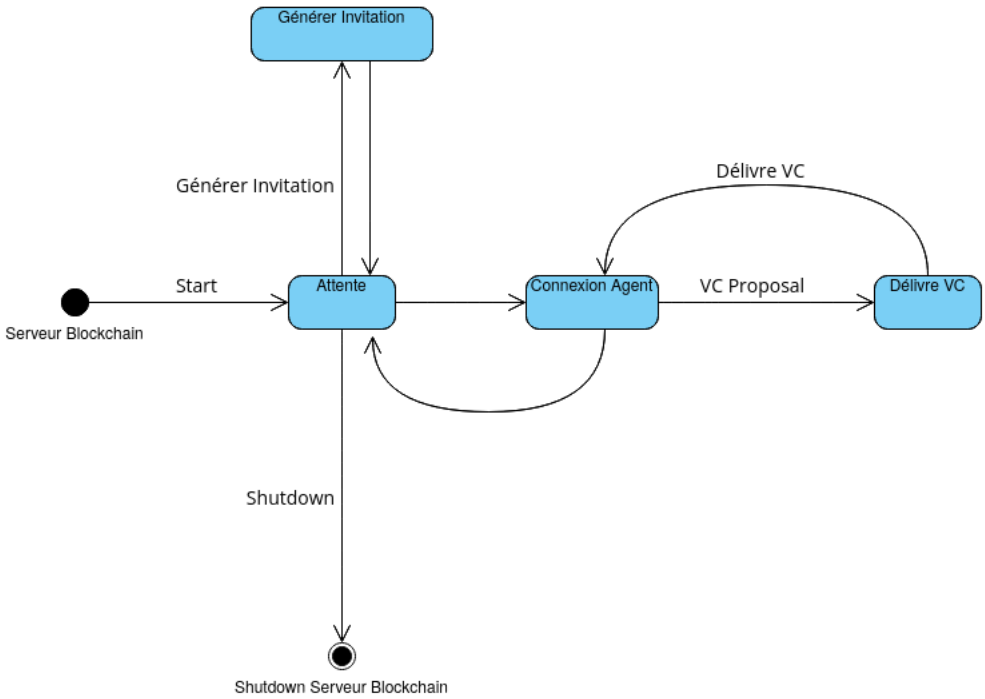


FIGURE 12 – Diagramme d'état de ServeurB

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 que 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 récupéré notre clé publique. Finalement, le VPN Wireguard peut être mis en place des deux côtés.
Nous avons mis en place une interface graphique pour pouvoir gérer les différentes intéractions et événements, ainsi qu'un diagramme d'états de l'application.

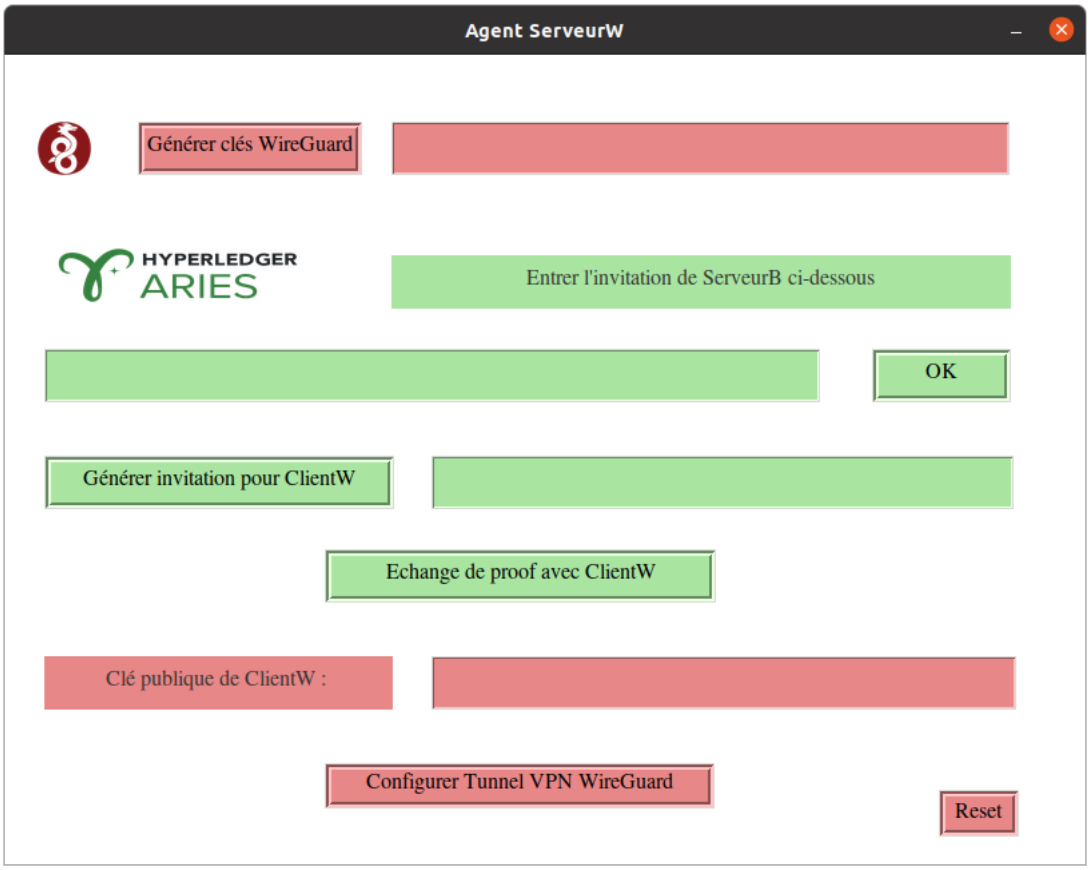


FIGURE 13 – Interface graphique de notre Serveur Wireguard

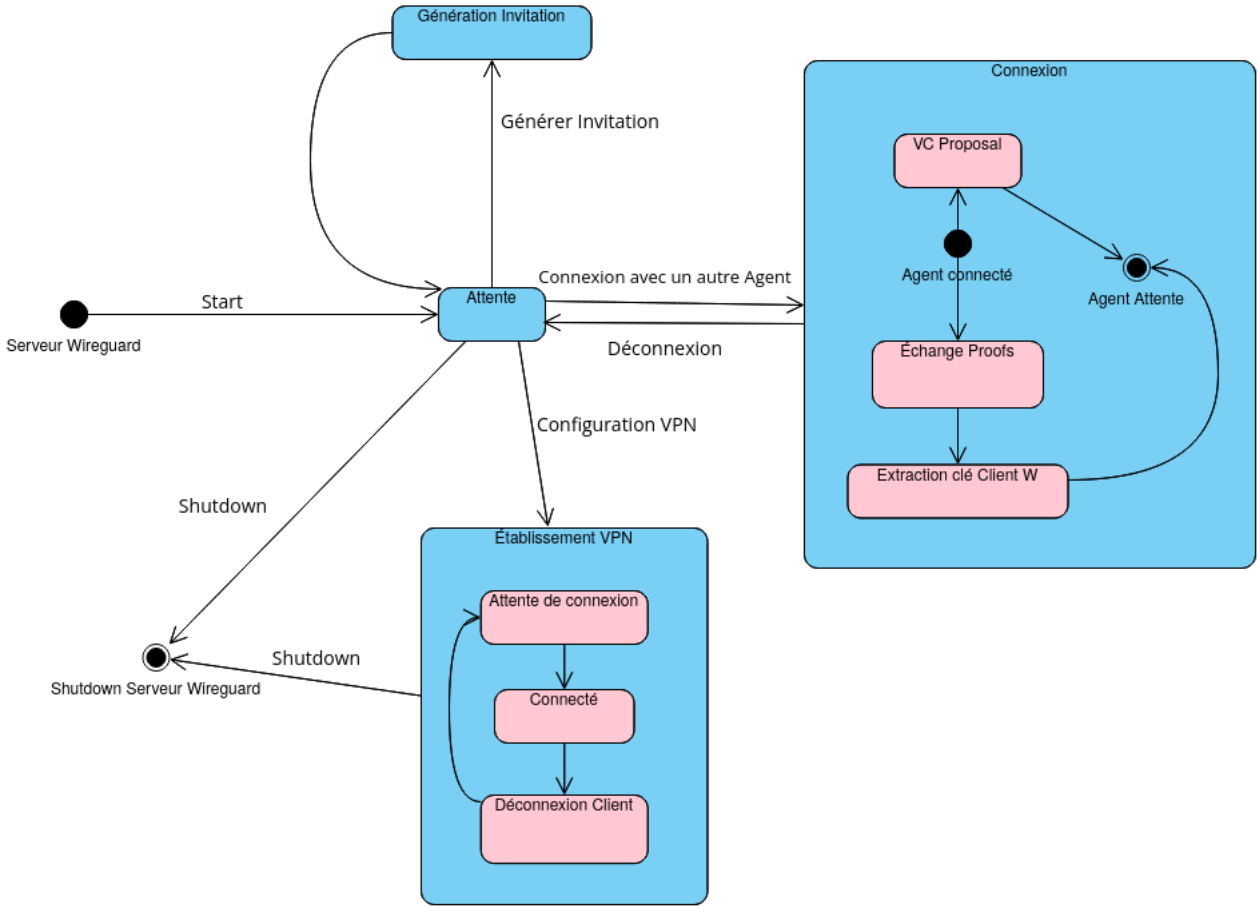


FIGURE 14 – Diagramme d'état de ServeurW

4.5 Agent du Client WireGuard

4.5.1 Mobile Agent

Le Mobile Agent initialement prévu repose sur l'application Aries Bifold. Nous pouvons voir ci-dessous le principe de son fonctionnement prévu à l'origine sous forme de diagramme UML de séquences :

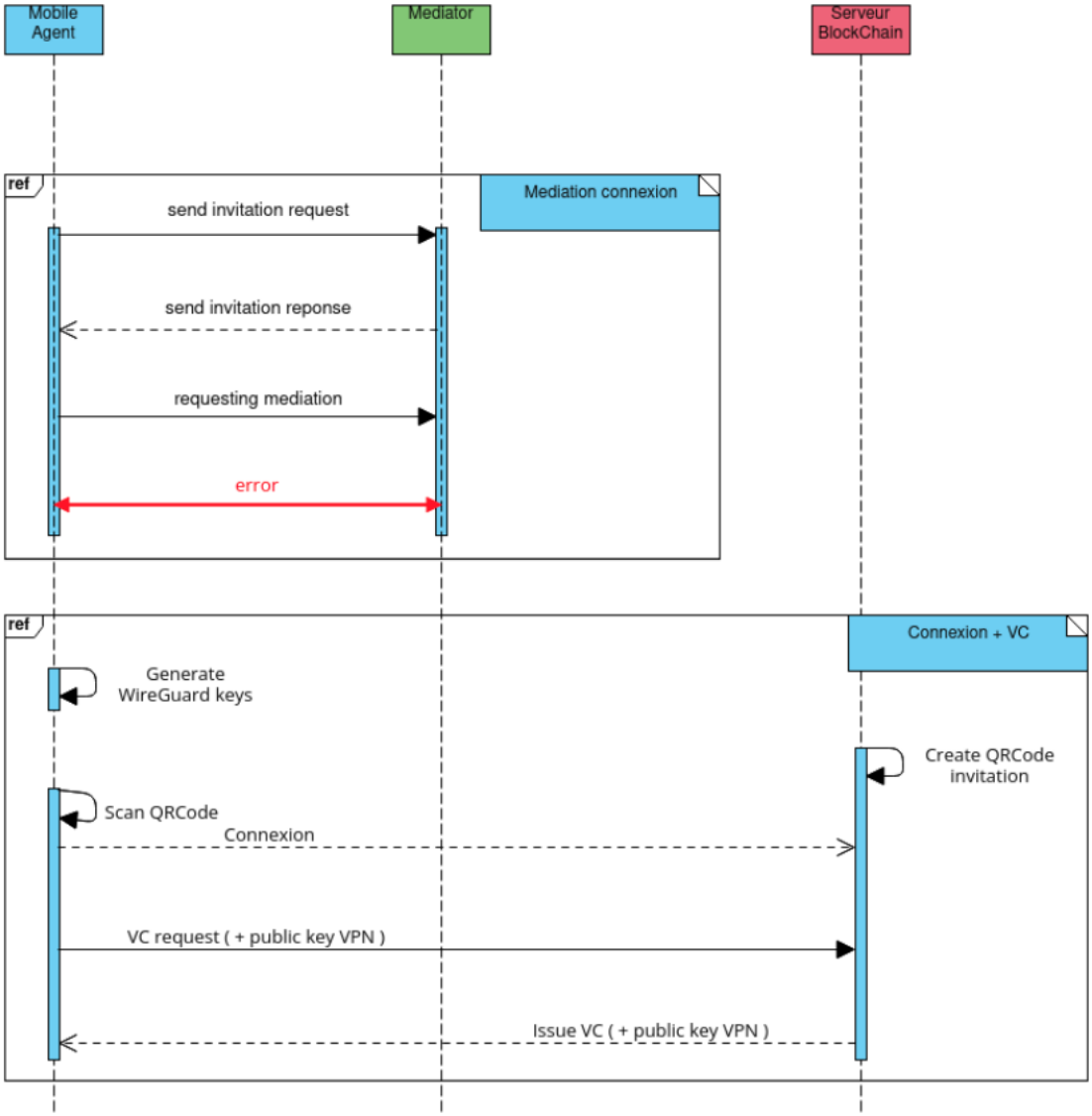


FIGURE 15 – Diagramme UML de séquence du Mobile Agent

Comme nous pouvons le voir ci-dessus, le Mobile Agent nécessite dans un premier temps d'établir une connexion avec un Mediator pour pouvoir ensuite par son intermédiaire discuter avec ServeurB et ServeurW. Par défaut, Aries Bifold propose l'utilisation d'un Mediator public clé en main basé sur ACA-py et développé par Indicio Technologies [4]. Malheureusement c'est au niveau du premier échange avec le Mediator que nous avons été bloqué. Nous avons alors essayé de nous-même concevoir notre propre Mediator en se basant sur les options de médiation proposées par Aries Cloud Agent Python. Ces options étaient ajoutées à l'Agent Cloud ServeurB, et elles nous ont aussi permis de voir comment les interactions et messages étaient reçus côté Mediator, chose qui n'était pas possible avec celui par défaut. Cependant nous nous sommes confrontés au même problème qu'avec celui par défaut.

Le Mobile Agent ne se contente pas d'effectuer des connexions DIDComm avec les Agents de ServeurB et de ServeurW, mais il doit aussi être capable de configurer le tunnel VPN WireGuard. L'application Aries Bifold par défaut ne propose que la connexion à l'aide de QRCode et la réception de VC. Nous avions donc prévu d'implémenter des services supplémentaires pour générer le couple de clés publiques/privées WireGuard, émettre des Proofs à ServeurW ainsi qu'établir le tunnel VPN une fois la clé publique de ServeurW

4. Indicio Public Mediator : `https://indicio-tech.github.io/mediator/`

extraite. On peut voir ci-dessous l'interface graphique de l'application (version Android apk) de Aries Bifold avec l'ajout des fonctionnalités dont on a besoin :

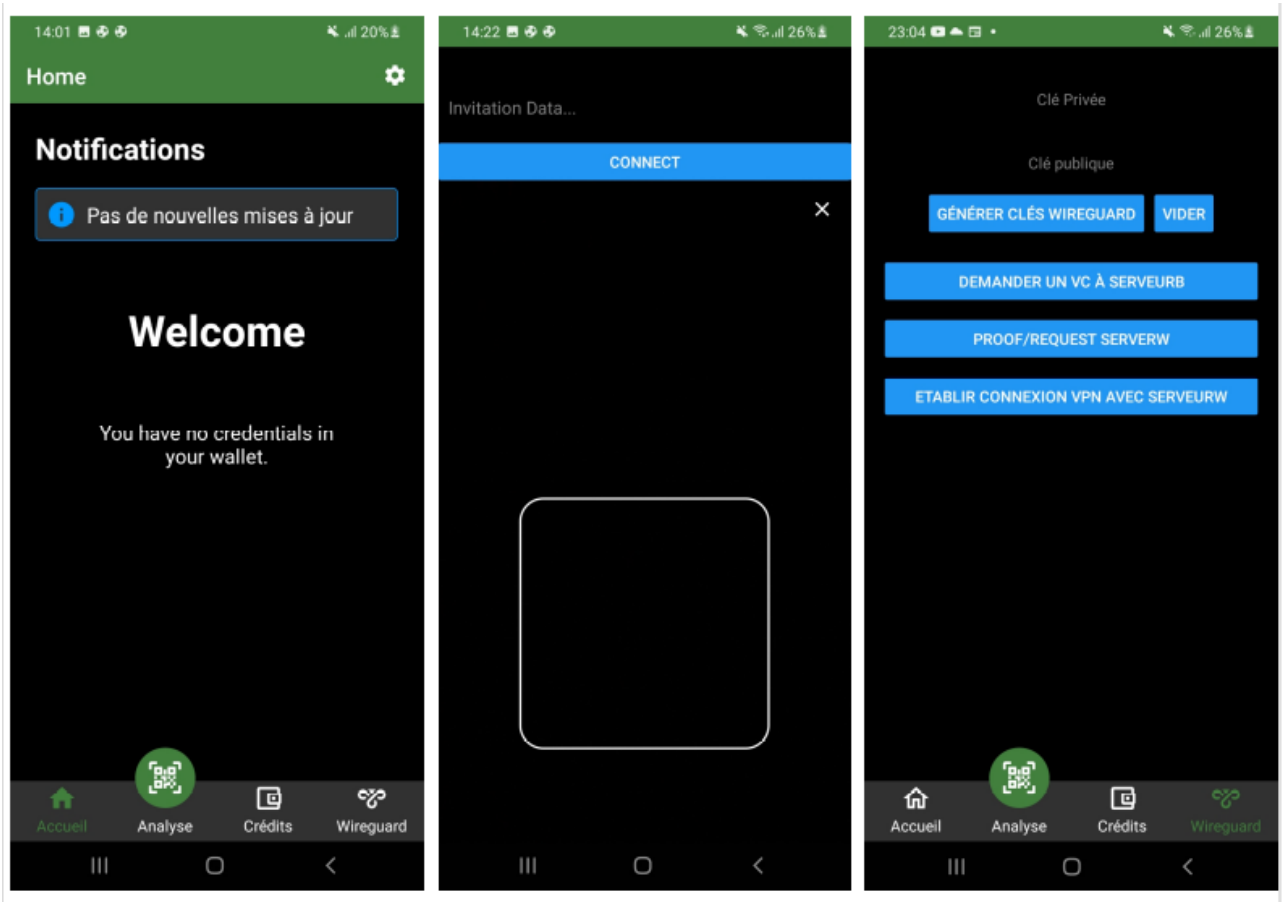


FIGURE 16 – GUI du Mobile Agent

Comme on peut le voir, nous avons ajouté un onglet WireGuard à l'application contenant toutes les options/étapes nécessaires sous forme de boutons cliquables. Nous avons également ajouté une option dans l'onglet Analyse pour établir la connexion en utilisant directement un URL en texte plutôt que via scannner de QRCode. cette option était nécessaire lorsqu'on utilisait l'application sur la VM ClientW Androidx86 car elle ne dispose pas d'objectif caméra.

Malheureusement, à cause des problèmes du Mediator et de l'impossibilité d'utiliser des messages DIDComm connexions et proofs en version 2 indispensables à nos Agents ServeurW et ClientW, nous nous sommes concentrés sur une version Aries Cloud-Agent Python sur Debian11 de ClientW et cette application Mobile Agent est restée sans suite. Nous avons fait ce choix car l'utilisation du Mobile Agent aurait nécessité de consacrer au développement d'un Mediator compatible, et cela n'était pas possible dans les délais.

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 contenait aussi sa clé publique, donc quand la Presentation est validée, le serveur et client Wireguard ont tous les deux les clés de l'autre. Ci-dessous nous pouvons voir l'interface graphique de l'Agent ClientW ainsi que le diagramme d'états associé :

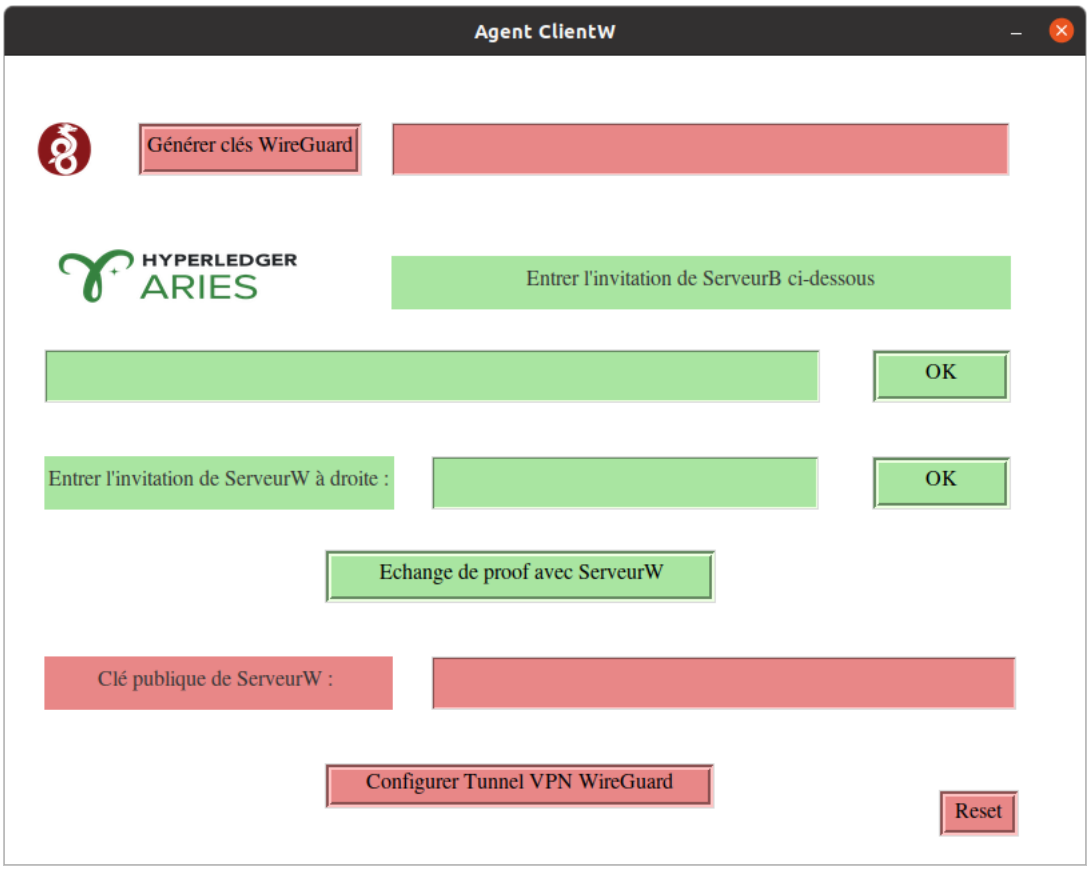


FIGURE 17 – Interface graphique de notre Client Wireguard

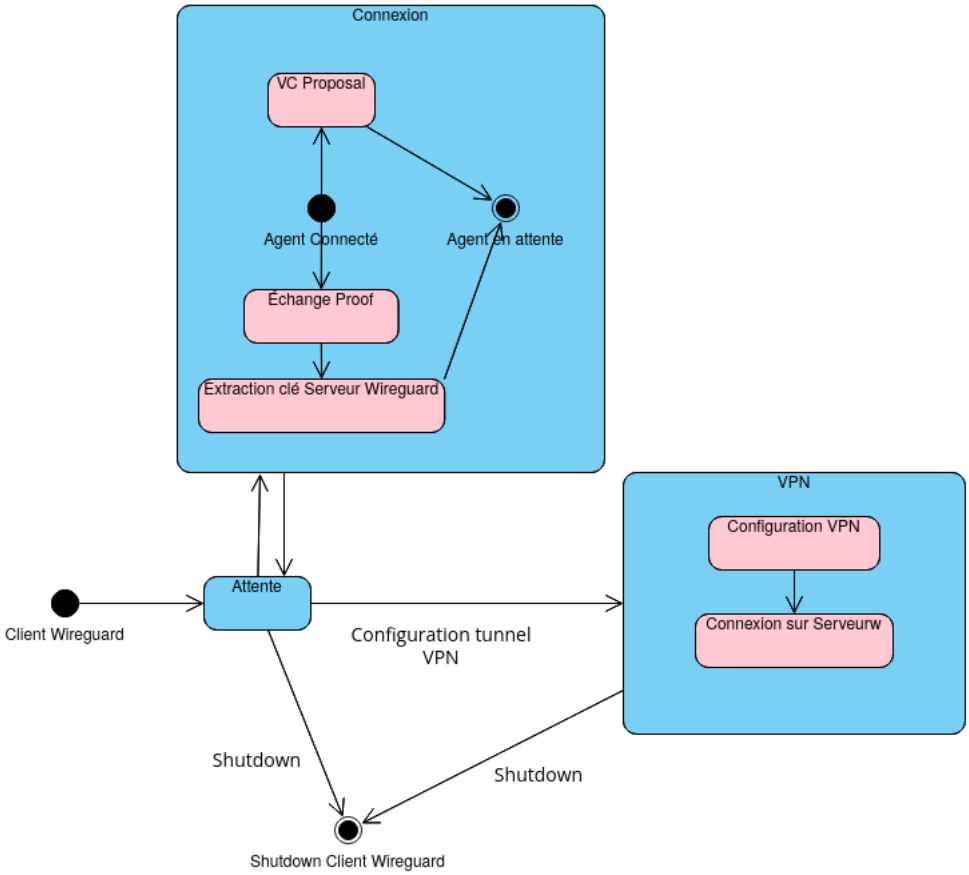


FIGURE 18 – Diagramme d'état de ClientW

5 Analyse du fonctionnement & Tests

5.1 Analyse du fonctionnement

Les différents projets Aries Hyperledgers utilisés pour la réalisation de ce projet étant en cours de développement et pour la plupart peu documentés, nous avons du réaliser de nombreux tests pour en analyser le fonctionnement et pour comprendre comment s'en servir avant de les implémenter à notre projet. Au niveau de Aries Cloud Agent Python, on retrouve des démos simplifiées basées sur des images Docker dans le dépôt officiel pour prendre en main les fonctionnalités. A l'aide de Wireshark nous observions le traffic entre les Agents des démos pour comprendre quels sont les échanges, les différents types de messages DIDComm apparaissant sous forme de requêtes http, etc.
Au niveau de Aries Mobile Agent React-Native, l'analyse du fonctionnement de l'application s'effectue au niveau d'Android Studio dans le terminal de logs. En combinant à Wireshark pour observer certains échanges non renseignés dans les logs, et pour comprendre quand des messages n'atteignent ou ne partent pas des agents, on peut finir par faire le lien avec la documentation théorique et sommaire du projet.
Tous les modules de notre projet ont un comportement attendu à la date de la remise, comme nous pourrons le voir ci-dessous avec des tests qui ont été effectués pour détecter, corriger et limiter les bugs ou erreurs majeurs. Bien entendu, une partie de notre code à savoir les Agents Aries de ServeurB, ClientW et ServeurW étant basés sur **Aries Cloud Agent - Python** et **von-network** sont voués à être rapidement obsolètes au vu de l'avancée des normes

DIDComm et Indy, ainsi que les modifications et mises à jour régulières de ce projet en cours de développement. Cela explique aussi pourquoi certains cas d'erreurs potentiels n'ont pas pu être testés de part la difficulté de les mettre en place avec cette base existante complexe en constante modification, surtout au niveau des échanges de messages DIDComm et des requêtes HTTP entre Agents Client/Serveur.

5.2 Tests

Les tests permettent de vérifier le bon fonctionnement d'une petite partie bien précise (module) de notre application. Ils s'assurent qu'une méthode exposée à la manipulation par un utilisateur fonctionne bien de la façon dont elle a été conçue. Les tests ont pour principal objectif de garantir une qualité de service dans des conditions réelles d'utilisation.

5.2.1 Test de couverture

Les tests de couverture ont pour principal objectif de mesurer la quantité de code couvert lors de l'exécution de tests. Cela permet aussi d'obtenir des informations sur les sections de code non testées.

Name	Stmts	Miss	Cover
/Users/cherifdiallo/Etude/Master/S10/PFE/src/Cloud-Agent/Front-end/QrCode_Generation.py	34	21	38%
/Users/cherifdiallo/Etude/Master/S10/PFE/src/Cloud-Agent/Front-end/clientW/clientW_Front.py	272	60	78%
/Users/cherifdiallo/Etude/Master/S10/PFE/src/Cloud-Agent/Front-end/serveurW/serveurW_Front.py	271	62	77%
test_clientW_Front.py	50	0	100%
test_serveurW_Front.py	52	0	100%
TOTAL	679	143	79%

FIGURE 19 – Resultat des tests de couverture

Les résultats présentés en figure 19 ci-dessus montrent que la moyenne de couverture de l'ensemble des tests sur les différentes parties de notre projet est de 79%.
Avoir une couverture de 100% ne veut rien dire, tout dépend du type de couverture utilisé. Chaque couverture a ses points forts, points faibles et son lot d'informations.
Le choix de la couverture dépend de ses besoins. Pour les tests vitaux une couverture de méthode de 100% peut sembler pertinente alors qu'elle ne l'est pas pour des tests de régression et encore moins des tests de validation. Pour ces tests nous avons fait des tests de couverture des méthodes (des fonctionnalités), cette couverture correspond à la couverture la plus basique. Elle correspond au pourcentage de fonctionnalités de test effectué.
Donc, dans notre cas on a fait des tests sur la fonctionnalité. Par exemple dans le cadre de la fonction loadJSON() dont le but est de lire le contenu d'un fichier JSON, il faut s'assurer de plusieurs choses qui sont :
- Est-ce que le fichier à lire existe ?
- Est-ce que la fonction lit le bon fichier ?
- Est-ce que la fonction retourne le contenu demandé ?

5.2.2 Discussion des résultats

Quand on observe les pourcentages obtenus par rapport aux différents codes que nous avons produit, on note que le ServerB est le moins couvert. Cela est dû au fait que ServeurB est plus

complexe à tester de part l'accessibilité difficile au VON-Network dockerisé, ainsi qu'à toute la partie BlockChain Indy qui est initialement un projet à part entière. Cela nous aurait pris beaucoup trop de temps et les délais ne permettaient pas de s'attarder dessus.

6 Conclusion

Le besoin de pouvoir prouver son identité auprès d'un serveur, et viceversa, peut-être reglé avec des certificats délivrés par une authorité à laquelle on fait confiance. Cependant, le fait que l'authorité soit celle qui délivre les certificats impose une dynamique de pouvoir qui peut devenir gênante pour l'autre partie devenant dépendante de l'authorité. Si l'authorité le décide, elle peut par exemple révoquer un certificat, ou alors si elle attaquée, tous les certificats qu'elle aura délivré seront compromis. Avec l'apparition des technologies BlockChain reposant sur la décentralisation, nous avons la possibilité de considérer des alternatives différentes aux certificats délivrés par une authorité.
L'objectif du projet *Déploiement d'un VPN sur des équipes mobiles ou IoT* était de mettre en place un VPN sur un serveur et de pouvoir avoir un client mobile qui s'y connecte. Cela nécessite donc une authentification et confiance mutuelle entre les deux hôtes lors de la mise en place du VPN par échange de clés publiques. Les certificats délivrés par authorité de certification peuvent être utiles dans ce cas, car ils permettraient d'ajouter un tier de confiance dans l'échange. Néanmoins, cette vision centralisée pose différents problèmes évoqué précédemment. L'option alternative choisie ici repose donc sur l'utilisation des technologies Blockchain, à savoir sur le projet Aries Hyperledger soutenu par la Fondation Linux. Cet ensemble de sous-projets permet la délivrance de certificats décentralisés ou Verifiable Credentials.
Au terme de ce projet nous avons pu découvrir le secteur novateur de la BlockChain, des normes de communication DIDComm ainsi que de l'ensemble du projet Aries Hyperledger. Malgré les nombreux problèmes auxquels nous nous sommes heurtés, nous avons tout de même réussi à mettre en place un réseau virtuel NEmu permettant de simuler les interactions entre 3 Aries Cloud Agent dans le but de délivrer des VC, les vérifier et établir un tunnel VPN dont les hôtes sont authentifiés par la BlockChain. Toutes ces étapes ont été regroupées dans trois applications pour chacun des agents avec des interfaces utilisateurs réalisées en Python Tkinter. En annexe 8.1 de ce rapport on peut voir le Diagramme de Gantt final au terme du projet. On peut y voir les 3 phases de l'avancée du projet correspondant aux scénarios représentés par les 3 couleurs.

6.1 Limitations

Le projet Hyperledger promet de proposer des librairies, outils et frameworks qui rendraient possible l'adaptation et utilisation des technologies Blockchain au sein des entreprises. Cependant, ce projet est toujours en cours de développement ce qui présente une des plus grandes limites de notre projet. Un des plus grands objectifs de ce projet était de pouvoir déployer un Agent Aries Mobile sur Android. Malheureusement, cela n'a pas été possible, comme nous avons pu l'expliquer précédemment dans les parties lui étant réservées. La contrainte de temps, avec la limite de développement du côté Agent Mobile nécessitant de développer en plus un Mediator, nous as contraint à abandonner cet agent pour nous concentrer sur un autre qui est lui fonctionnel, le Cloud Agent.
A l'origine de ce projet, nous voulions déployer différents types d'agents. Un Agent Cloud pour le serveur Blockchain, un Agent Cloud pour le serveur VPN et un Agent Mobile pour le

client VPN. L'Agent Mobile devait lui-même à l'origine être un Static-Agent malheureusement non compatible avec le projet. Il aurait pu être intéressant de mettre en lien différents types d'agents sur des plateformes différentes pour comparer leurs modes de fonctionnement, leurs compatibilités, etc.
Ce projet déploie des Agent Aries Cloud sur des machines Debian11. Ces Agents sont en communication avec un réseau de noeuds Blockchain, dans notre cas un VON-Network dockerisé. Il y a des tests que nous avons pu faire et que nous avons détaillé dans la partie Tests, mais il y en a que malheureusement nous n'avons pas pu mettre en place, de part la complexité de tester toutes les interactions entre les différents serveurs interrogés de manière indirecte par requêtes HTTP POST et GET. Toute la partie VON-Network initialement prévue dans le Projet 2 n'a également pas pu être testée par manque de temps. De nombreux cas d'erreurs non traités à notre niveau dans les applications sont cependant affichés dans les terminaux des agents par des retours d'erreurs des serveurs aca-py. Un traitement des sorties STDOUT des processus pourrait être mis en place pour à minima renseigner les erreurs proprement dans les interfaces graphiques des agents.
Pour finir, ce projet étant dépendant de l'instabilité et de la complexité de Aries Hyperledger Cloud Agent Python ainsi que de VON-Network, nous n'avons pas eu le temps de mettre en place des systèmes pour sécuriser nos Agents, comme par exemple le fait que les logs, les VC et informations relatives aux connexions soient enregistrés en clair dans des fichiers json dans le dossier *ressources/* par souci de praticité et de compréhension des informations. Ce projet n'est donc pas adapté à être mis en production.

6.2 Extensions

Une idée intéressante, et sûrement utile dans le futur serait de pouvoir exporter ce projet sur d'autres plateformes comme par exemple sur Raspberry Pi. L'intérêt ici serait son utilisation en IoT[5]. Un VPN pour IOT peut être utile dans certains cas, avec pour avantage de le combiner à la BlockChain et aux VC pour l'authentification. Raspbian étant un système proche de Debian sur lequel nous avons réalisé ce projet, cela pourrait faciliter le portage sous Raspberry.
Dans notre projet nous avons un client unique, mais dans la réalité il se peut que notre serveur VPN supporte plusieurs clients en simultané. WireGuard est en mesure de supporter plusieurs clients, une extension pourrait donc permettre l'accès au serveur pour plusieurs clients.
Une autre extension que nous avons commencé à implémenter mais non terminée par manque de temps concerne un bouton pour réinitialiser toutes les connexions DIDComm , les configurations VPN et les VC enregistrés par un agent (révocation de VC). Cela peut avoir un intérêt dans le cadre de problèmes techniques nécessitant une réinitialisation du système. On pourrait aussi ajouter côté client un bouton pour permettre de stopper et de réactiver le VPN sans forcément fermer complètement l'Agent.
Une dernière extension possible est de pouvoir révoquer des Verifiable Credentials sans pour autant réinitialiser tout le système via le bouton reset.

5. IOT : Internet Of Things, ou Objets connectés.

7 Bibliographie

— WireGuard : https://www.wireguard.com/ (consulté le 06/03/2022)

— World Wide Web Consortium (W3C) : https://www.w3.org/ (consulté le 06/03/2022)

— Verifiable Credentials : https://www.w3.org/TR/vc-data-model/ (consulté le 06/03/2022)

— Hyperledger : https://github.com/hyperledger (consulté le 15/03/2022)

— Aries Cloud Agent : https://github.com/hyperledger/aries-cloudagent-python (consulté le 15/03/2022)

— Aries Mobile Agent React-Native : https://github.com/hyperledger/aries-mobile-agent-react-native (consulté le 15/03/2022)

— QEMU : https://www.qemu.org/ , https://wiki.qemu.org/Main_Page (consulté le 15/03/2022)

— NEmu : https://gitlab.com/v-a/nemu (consulté le 15/03/2022)

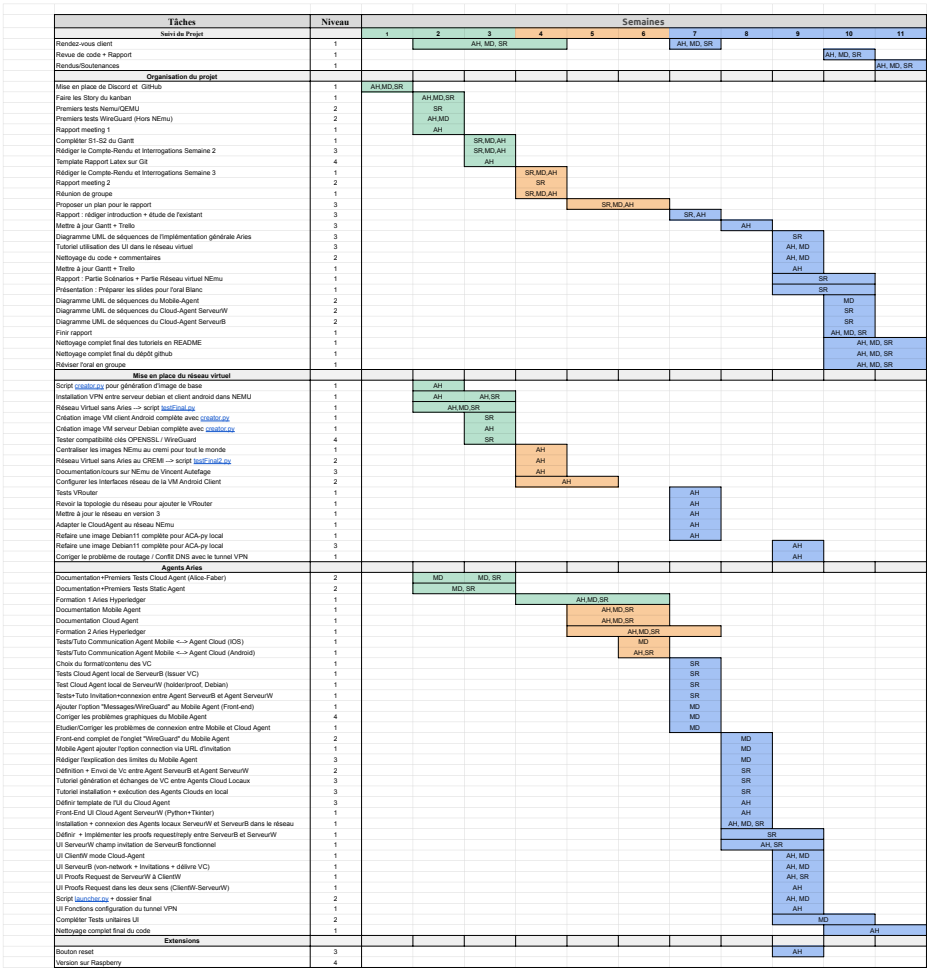
— Indicio Public Mediator : https://indicio-tech.github.io/mediator/ (consulté le 24/03/2022)

— VON-Network : https://github.com/bcgov/von-network (consulté le 24/03/2022)

— GreenLight Ledger : http://greenlight.bcovrin.vonx.io/ (consulté le 24/03/2022)

8 Annexes

8.1 Gantt Final du Projet



Tâches	Niveau	Semaines 1	2	3	4	5	6	7	8	9	10	11
Suivi de Projet												
Rendez-vous client	1		AH, MD, SR					AH, MD, SR				
Revue de code + Rapport	1										AH, MD, SR	
Rendus/Soutenances	1											AH, MD, SR
Organisation du projet												
Mise en place de Discord et GitHub	1	AH,MD,SR										
Faire les Story du kanban	1		AH,MD,SR									
Premiers tests Nemu/QEMU	2		SR									
Premiers tests WireGuard (Hors NEmu)	2		AH,MD									
Rapport meeting 1	1		AH									
Compléter S1-S2 du Gantt	1			SR,MD,AH								
Rédiger le Compte-Rendu et Interrogations Semaine 2	3			SR,MD,AH								
Template Rapport Latex sur Git	4			AH								
Rédiger le Compte-Rendu et Interrogations Semaine 3	1				SR,MD,AH							
Rapport meeting 2	2				SR							
Réunion de groupe	1				SR,MD,AH							
Proposer un plan pour le rapport	3					SR,MD,AH						
Rapport : rédiger introduction + étude de l'existant	3							SR, AH				
Mettre à jour Gantt + Trello	3								AH			
Diagramme UML de séquences de l'implémentation générale Aries	3									SR		
Tutoriel utilisation des UI dans le réseau virtuel	3									AH, MD		
Nettoyage du code + commentaires	2									AH, MD		
Mettre à jour Gantt + Trello	1									AH		
Rapport : Partie Scénarios + Partie Réseau virtuel NEmu	1									SR		
Présentation : Préparer les slides pour l'oral Blanc	1									SR		
Diagramme UML de séquences du Mobile-Agent	2										MD	
Diagramme UML de séquences du Cloud-Agent ServeurW	2										SR	
Diagramme UML de séquences du Cloud-Agent ServeurB	2										SR	
Finir rapport	1										AH, MD, SR	
Nettoyage complet final des tutoriels en README	1											AH, MD, SR
Nettoyage complet final du dépôt github	1											AH, MD, SR
Réviser l'oral en groupe	1											AH, MD, SR
Mise en place du réseau virtuel												
Script creator.py pour génération d'image de base	1		AH									
Installation VPN entre serveur debian et client android dans NEMU	1		AH	AH,SR								
Réseau Virtuel sans Aries --> script testFinal.py	1		AH,MD,SR									
Création image VM client Android complète avec creator.py	1			SR								
Création image VM serveur Debian complète avec creator.py	1			AH								
Tester compatibilité clés OPENSSL / WireGuard	4			SR								
Centraliser les images NEmu au cremi pour tout le monde	1				AH							
Réseau Virtuel sans Aries au CREMI --> script testFinal2.py	2				AH							
Documentation/cours sur NEmu de Vincent Aubefage	3				AH							
Configurer les Interfaces réseau de la VM Android Client	2					AH						
Tests VRouter	1							AH				
Revoir la topologie du réseau pour ajouter le VRouter	1							AH				
Mettre à jour le réseau en version 3	1							AH				
Adapter le CloudAgent au réseau NEmu	1							AH				
Refaire une image Debian11 complète pour ACA-py local	1							AH				
Refaire une image Debian11 complète pour ACA-py local	3									AH		
Corriger le problème de routage / Conflit DNS avec le tunnel VPN	1									AH		
Agents Aries												
Documentation+Premiers Tests Cloud Agent (Alice-Faber)	2		MD	MD, SR								
Documentation+Premiers Tests Static Agent	2		MD, SR									
Formation 1 Aries Hyperledger	1				AH,MD,SR							
Documentation Mobile Agent	1					AH,MD,SR						
Documentation Cloud Agent	1					AH,MD,SR						
Formation 2 Aries Hyperledger	1					AH,MD,SR						
Tests/Tuto Communication Agent Mobile <--> Agent Cloud (IOS)	1						MD					
Tests/Tuto Communication Agent Mobile <--> Agent Cloud (Android)	1						AH,SR					
Choix du format/contenu des VC	1							SR				
Tests Cloud Agent local de ServeurB (Issuer VC)	1							SR				
Test Cloud Agent local de ServeurW (holder/proof, Debian)	1							SR				
Tests+Tuto Invitation+connexion entre Agent ServeurB et Agent ServeurW	1							SR				
Ajouter l'option "Messages/WireGuard" au Mobile Agent (Front-end)	1							MD				
Corriger les problèmes graphiques du Mobile Agent	4							MD				
Etudier/Corriger les problèmes de connexion entre Mobile et Cloud Agent	1							MD				
Front-end complet de l'onglet "WireGuard" du Mobile Agent	2								MD			
Mobile Agent ajouter l'option connection via URL d'invitation	1								MD			
Rédiger l'explication des limites du Mobile Agent	3								MD			
Définition + Envoi de Vc entre Agent ServeurB et Agent ServeurW	2								SR			
Tutoriel génération et échanges de VC entre Agents Cloud Locaux	3								SR			
Tutoriel installation + exécution des Agents Clouds en local	3								SR			
Définir template de l'UI du Cloud Agent	3								AH			
Front-End UI Cloud Agent ServeurW (Python+Tkinter)	1								AH			
Installation + connexion des Agents locaux ServeurW et ServeurB dans le réseau	1								AH, MD, SR			
Définir + implémenter les proofs request/reply entre ServeurB et ServeurW	1								SR			
UI ServeurW champ invitation de ServeurB fonctionnel	1								AH, SR			
UI ClientW mode Cloud-Agent	1									AH, MD		
UI ServeurB (non-network + Invitations + délivre VC)	1									AH, MD		
UI Proofs Request de ServeurW à ClientW	1									AH, SR		
UI Proofs Request dans les deux sens (ClientW-ServeurW)	1									AH		
Script launcher.py + dossier final	2									AH, MD		
UI Fonctions configuration du tunnel VPN	1									AH		
Compléter Tests unitaires UI	2										MD	
Nettoyage complet final du code	1											AH
Extensions												
Bouton reset	3									AH		
Version sur Raspberry	4											