Gestionnaire de mots de passe décentralisé

Rapport final PX511

Sommaire

I) Introduction

Avec un nombre croissant de comptes à créer sur Internet pour pouvoir avoir accès à tel ou tel service, se pose la problématique des mots de passe : respecter les bonnes pratiques recommandées par l'ANSSI[1] pour trois ou quatre mots de passe est faisable, mais d'après NordPass[2], une personne moyenne aurait environ 70 mots de passe actifs. Nous sommes alors tous tentés de réutiliser les mêmes mots de passe pour plusieurs comptes : d'après une enquête d'Avast[3] en 2019, 93% des français ne respectent pas les bonnes pratiques en termes de mots de passe. Une solution a émergé pour répondre à ce problème : les gestionnaires de mots de passe. À partir d'un seul mot de passe maître (qui doit, celui-ci, être parfaitement sécurisé), il est possible d'accéder à tous ses mots de passe sans les retenir, mais aussi de créer des mots de passe sécurisés. Plus besoin de créer ses

mots de passe, plus besoin de les retenir, tout cela de manière bien plus sécurisée qu'avant, les gestionnaires semblent la solution idéale.

Aujourd'hui, les gestionnaires de mot de passe connaissent un essor certain : il y a une prise de conscience générale de leur apport en termes de sécurité. Néanmoins, leur utilisation reste parfois obscure au grand public, les solutions sont souvent payantes car elles passent par un tiers de confiance, ou ne présentent pas de synchronisation des mots de passe entre différents appareils.

II) Définition des objectifs de l'étude théorique

L'objectif final est de trouver une solution gérant les mots de passe de manière simple (accessible au plus grand nombre), décentralisée, et assurant la synchronisation.
Précisons ce que nous entendons par "synchronisation" et "décentralisation".

Étant donnée une liste de mots de passe stockée sur un de nos appareils, nous devons pouvoir les partager à d'autres appareils (du même utilisateur) afin qu'ils puissent également les stocker et que nous puissions accéder à nos mots de passe depuis ces autres appareils. Ainsi, si nous créons un mot de passe sur un appareil, nous pourrons mettre à jour la liste de mots de passe sur un deuxième appareil afin de pouvoir utiliser ce nouveau mot de passe : c'est la synchronisation.

De plus, nous souhaitons éviter la centralisation des mots de passe, c'est-à-dire que pour accéder aux mots de passe, les appareils ne devront pas interroger une unique entité qui leur délivrera le mot de passe demandé (traditionnel modèle client-serveur). Cela pour éviter le coût que peut représenter un serveur, qui rendrait la solution payante et donc restreindrait l'accès au grand public.

Ainsi, nous ferons en premier lieu une brève étude du fonctionnement général d'un gestionnaire de mots de passe actuel, afin d'en dégager les problématiques et de mieux appréhender leurs fonctionnalités.

Ensuite, nous établirons une liste exhaustive des solutions de communication entre deux appareils (pour partager les mots de passe), nous définirons les avantages et les défauts de chacune de ces solutions afin de les comparer et de dégager celles qui semblent le plus adaptées à notre solution. Nous nous attarderons sur leur capacité à effectuer la synchronisation de la manière la plus efficace possible, ainsi qu'à respecter la décentralisation.

III) État de l'art

1) Fonctionnement d'un gestionnaire

De manière générale, un gestionnaire de mots de passe fonctionne de la manière suivante : un mot de passe maître est créé en premier. Ensuite, pour s'authentifier auprès du

[1][1]https://www.ssi.gouv.fr/guide/mot-de-passe/
[2]https://www.newswire.com/news/new-research-most-people-have-70-80-passwords-21103705
[3]https://press.avast.com/fr-fr/enqu%C3%AAte-avast-93-des-fran%C3%A7ais-utilisent-des-mots-de-passe-faibles

serveur (afin de créer un mot de passe ou bien d'accéder à un de vos services), il faudra saisir ce mot de passe maître. Tous vos mots de passe sont stockés au même endroit (dans la base de données du serveur). Quand vous voulez accéder à vos mots de passe, vous saisissez le mot de passe maître, qui est ensuite dérivé deux fois de manière différente, une fois pour obtenir la clef qui servira à déverrouiller le coffre-fort contenant vos mots de passe (cette clef n'est jamais envoyée au serveur, il n'y a que vous qui la possédez), et une autre fois pour obtenir la clef qui permettra de vous authentifier auprès du serveur, qui vous délivrera alors le coffre-fort contenant vos mots de passe. Ainsi, même si le coffre-fort est intercepté, il ne pourra être déverrouillé car seul vous possédez la clef, qui n'est jamais envoyée sur le réseau.

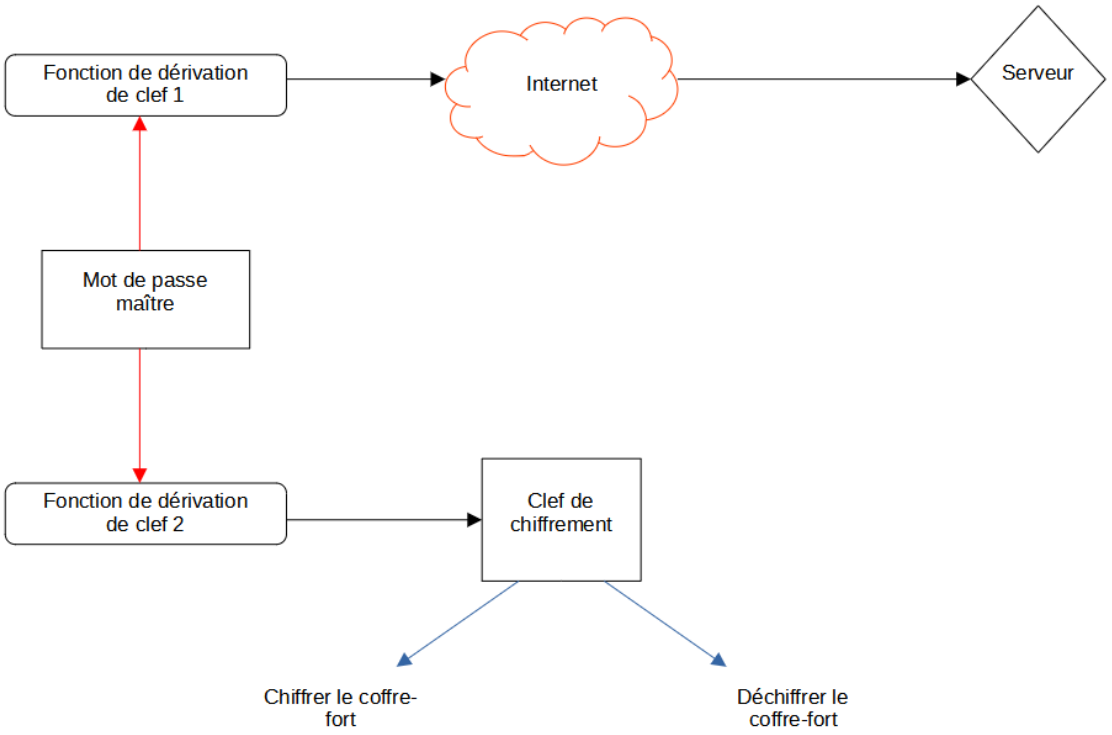


figure 1 : Résumé du fonctionnement général d'un gestionnaire de mots de passe

2) Les solutions de communication

Dans cette partie, nous allons détailler les différentes solutions de communication entre deux appareils qui permettent de partager les mots de passe.

Nous ferons une distinction entre les solutions utilisables localement, et celles qui permettent une synchronisation à distance :

- La **technologie Bluetooth**, permettant une communication à courte distance entre les appareils et permettant le transfert de données.

- L'utilisation de **QR Code** a aussi été prise en compte, permettant d'obtenir le lien pour la réception du fichier à partager. Le transfert ne peut cependant se faire que dans le sens "PC vers smartphone/tablette" et non "smartphone/tablette vers PC", du fait de la nécessité d'un scanner QR Code.

- La synchronisation **grâce à un support,** comme une clef usb ou une carte SD : l'appareil source met les données sur le support, on met le support sur l'appareil destination et on les récupère. On peut aussi le faire à l'aide d'un câble. Cette solution est très simple mais relativement contraignante pour l'utilisateur.

Un autre type de solution prend en compte le fait que les appareils à synchroniser soient distants. Pour ce cas de figure, nous avons dégagé différentes possibilités :

- L'utilisation du **protocole SFTP** (Secure File Transfer Protocol) peut se décliner en deux solutions. Soit nous utilisons l'appareil source comme un serveur pour envoyer au deuxième appareil, mais dans ce cas, il faudra faire une configuration NAT car il y a un trafic entrant, soit nous utilisons un serveur sftp dans le cloud depuis lequel les informations seraient rapatriées à la demande d'un utilisateur, mais cela nécessite encore un serveur.

- Le modèle de réseau **Peer to Peer** permettrait également de résoudre le problème de synchronisation : afin de récupérer le fichier cible, l'appareil va interagir avec les noeud du réseau qui possèdent une partie du fichier en interrogeant le serveur possédant la liste des noeuds et les fichiers qu'ils possèdent. Bien que ce modèle passe par l'utilisation d'un serveur pour guider les appareils vers le fichier cible, sa charge de travail reste réduite et peut sembler viable comme solution pour notre projet.

- La **Blockchain** est un modèle décentralisé qui se popularise de plus en plus dans différents domaines. Elle permet un partage de données décentralisé entre les nœuds qui contribuent à cette Blockchain. Les utilisateurs peuvent y stocker des données en les envoyant à un nœud qui va les vérifier puis les ajouter à la Blockchain. Tous les blocs de la Blockchain sont partagés avec l'ensemble des nœuds. Ce type de technologie présente malgré tout certains problèmes : la liste de mots de passe (dans notre cas), bien que chiffrée de façon rigoureuse, sera toujours présente sur la Blockchain, disponible par tous par définition même du principe de la Blockchain. Cela implique qu'elle sera à la merci d'individus pouvant tenter de décrypter nos informations (du fait que le fichier chiffré sera toujours disponible pour eux). La mise à jour de n'importe quel mot de passe de notre liste produira également à chaque fois un nouveau fichier chiffré de l'ensemble de nos mots de passe sur la Blockchain, toujours non supprimable. De plus, le principe de non-répudiation fait que le système garde une trace de qui a ajouté une information sur la Blockchain, laissant ainsi la gestion de la sécurité de notre anonymat dans leurs mains.

- L'utilisation du **protocole IPv6** est une voie possible pour réaliser la communication entre nos appareils distants. Le fait de passer par ce protocole permet une communication directe sans passer par le NAT, de par le nombre d'adresses disponibles. Cependant, si techniquement cela résout le problème, en réalité les NAT sont présents partout dans le monde et les adresses IPV6 sont parfois bloquées par les firewall, on ne peut donc pas utiliser cette solution dans tous les cas, en fonction de l'utilisateur.

- Les **solutions "stateless**" fonctionnent différemment : cette solution va combiner le mot de passe maître de l'utilisateur et les informations du site sur lequel on veut se connecter (nom du site, URL,..) pour créer le mot de passe qui sera utilisé. Cela demande cependant de modifier l'ensemble de ses mots de passe utilisant ce type de solution et aura des limites dans le cas où l'on doit nécessairement utiliser le même mot de passe pour des sites différents, où le cas de sites nécessitant un mode d'authentification spécifique, notamment ceux demandant un format de mot de passe particulier (chiffres uniquement ou caractères spéciaux interdits). Dans ces cas-là, la solution stateless ne sera pas utilisable. De plus, si on doit changer le mot de passe maître, il est nécessaire de changer tous les mots de passe.

- Une solution basée sur **le protocole ICE(RFC5245)**. L'idée est d'utiliser ce protocole, couplé aux protocoles STUN(RFC5389) et TURN(RFC5766), afin que chaque appareil puisse découvrir si il se trouve derrière un NAT (dans le cas contraire on peut communiquer directement sans avoir de soucis liés au NAT), et si c'est le cas, leur fournir des adresses qu'ils pourront utiliser pour communiquer de pair à pair.

En premier lieu, chaque appareil interroge un serveur STUN/TURN, qui informera le client de s'il se trouve derrière un NAT, et si c'est le cas lui fournira deux adresses publiques utilisables, l'adresse du NAT le plus éloigné du client (i.e le NAT a priori visible sur internet), et une adresse publique du serveur. Le client dispose alors a priori de trois adresses utilisables : sa propre adresse IP, celle du NAT, et celle du serveur.

Ensuite, le couple d'adresse qui sera utilisé pour communiquer directement entre les appareils est négocié par le protocole ICE. En effet, parmi les six adresses en jeu, certaines ne pourront pas fonctionner dans tous les cas (les adresses IP des appareils seront des adresses privées s' ils sont situés derrière des NAT et ne pourront pas fonctionner par exemple). Une fois ceci fait, une connexion directe s'établit pour l'échange des mots de passe.

3) Conclusion sur les solutions les plus adaptées

Afin de pouvoir établir une comparaison entre les différentes solutions potentielles, nous avons défini certains critères tels que :

- la synchronisation locale, permettant de partager un fichier entre appareils proches
- la synchronisation distante

- la synchronisation facile, avec un ensemble d'actions simples à réaliser côté utilisateur
- l'utilisation d'une technologie accessible pour l'utilisateur, ne demandant pas ou peu de configuration manuelle pour l'utilisateur
- des échanges de données sécurisés, c'est-à-dire limitant la présence de nos mots de passe (même bien chiffrés) sur le réseau.

	Synchronisation locale	Synchronisation distante	Synchronisation facile	Technologie accessible pour l'utilisateur	Echange des données "sécurisé"
Réseau local					
Clé USB					
Bluetooth					
QrCode					
StateLess					
SFTP					
P2P					
BlockChain					
IPV6					
ICE					

Précisons que les solutions suivantes ne sont pas complètement à mettre de côté, elles présentent chacune des avantages non négligeables et pourraient être utiles dans certaines circonstances. Cependant, dans le cadre de ce projet, nous devons choisir les solutions les plus adaptées, c'est pourquoi il est nécessaire de faire une sélection appropriée.

Solutions écartées:

- Tout d'abord, les solutions qui ne permettent pas d'avoir un échange de données très sécurisé (sans accès aux données lors de la communication ou plus tard) sont écartées. Ainsi la BlockChain qui conserve les données éternellement (donc nos mots de passe chiffrés) ne peut pas être viable pour un projet de gestionnaire de mot de passe. La technologie Peer to Peer pourrait sembler viable pour notre projet, cependant, il se pose des questions autour de la configuration du NAT de la part du client Peer to Peer (client torrent par exemple) qui ne sont pas résolues.
- Ensuite les solutions difficiles d'utilisation comme SFTP qui nécessitent une configuration sur les appareils pour utiliser le protocole et peut nécessiter de passer par du NAT pour pouvoir fonctionner, rendent la mise en œuvre compliquée pour le grand public. On l'écarte donc.

La solution IPV6, quant à elle, n'est pas accessible par un grand nombre de personnes pour l'instant donc on l'écarte aussi.

- La solution StateLess est intéressante mais la synchronisation reste difficile sans serveur. On peut l'écarter.
- La solution de l'utilisation du réseau local pour synchroniser des appareils est très intéressante pour synchroniser des appareils qui sont sur le même réseau. Cependant, l'objectif ici sur ce projet est de trouver des alternatives à cette solution donc on ne s'y penchera pas.
- L'utilisation d'une clé USB ou autre support pour synchroniser des appareils est efficace mais contraignant pour l'utilisateur. De plus, l'intérêt dans ce projet de s'intéresser à ce type de synchronisation est assez bas donc on écarte cette solution.

Solutions retenues:

Nous allons retenir plusieurs solutions pour que l'utilisateur puisse utiliser la solution la plus adaptée à son cas de figure.

- QrCode:

  L'utilisateur va pouvoir synchroniser facilement ses mots de passe depuis un appareil (PC, smartphone, tablette) vers un appareil portable (smartphone ou tablette).

  Cette solution est simple pour synchroniser 2 appareils proches.
- Bluetooth:

  L'utilisateur va pouvoir synchroniser facilement ses mots de passe entre 2 appareils quels qu'ils soient (possédant quand même le bluetooth) à proximité.
- ICE:

  La solution a été retenue car elle permet une synchronisation à distance facile, de manière décentralisée car elle permet une communication pair à pair, elle remplit donc a priori tous les objectifs que nous nous sommes fixés.

IV) Définition des objectifs de l'étude expérimentale

Pour tester ces solutions envisagées, nous allons développer un programme qui va pouvoir synchroniser des mots de passe entre 2 appareils. Le but est qu'un appareil puisse envoyer des mots de passe chiffrés via les solutions retenues (QrCode, Bluetooth, ICE) et qu'un second reçoive ces données.

Nous allons alors réaliser une solution commune avec 3 modes (un pour chaque moyen de communication). Cette solution aura plusieurs fonctionnalités :

- Recherche d'un appareil pour communiquer.
- Envoyer des données.
- Recevoir des données.

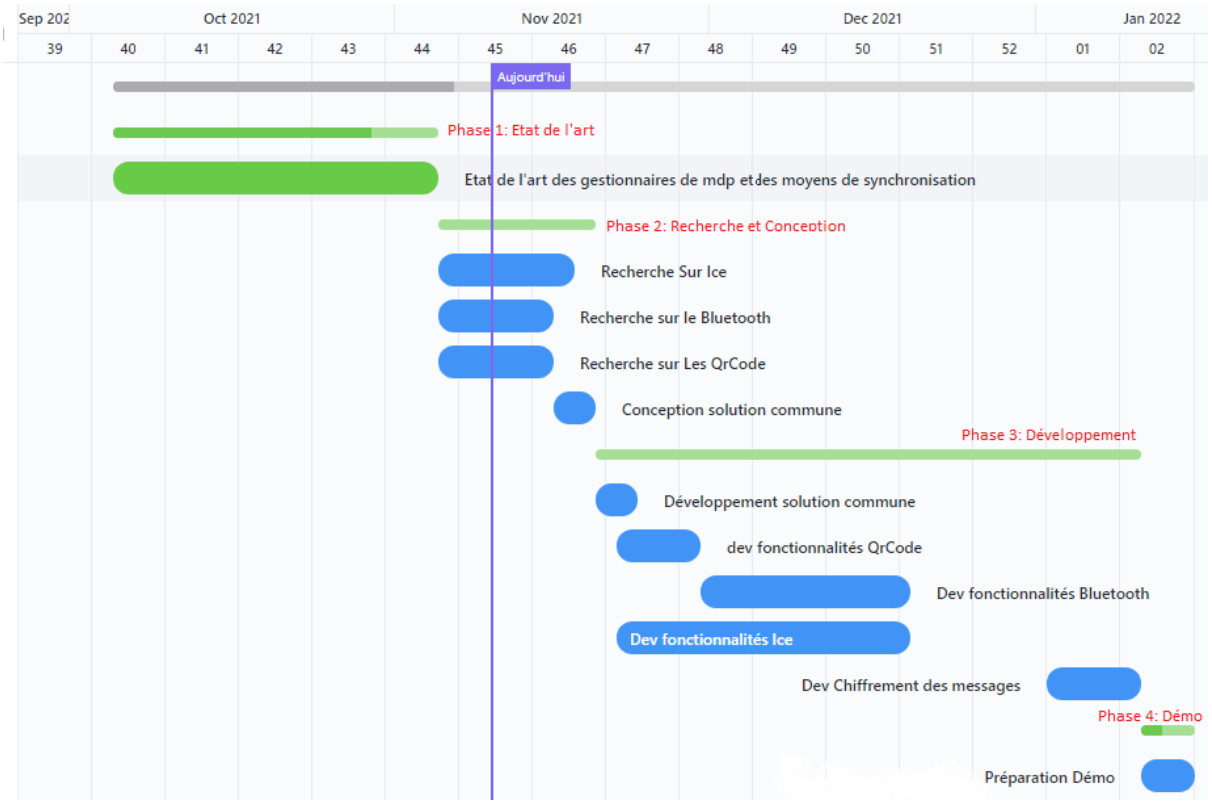
Ces fonctionnalités seront disponibles pour les 3 modes que nous pourrons choisir au départ. Une solution commune aux 3 modes sera utile pour d'une part pour harmoniser

notre code et d'autre part, si nous voulons rajouter des modes supplémentaires, il sera plus simple de les intégrer. De plus, le but est de faciliter l'utilisation par l'utilisateur qui n'aura qu'à choisir en fonction de son besoin, simplement en cliquant sur un bouton pour choisir son mode par exemple.

Pour finir, après que les différentes fonctionnalités soient prêtes, nous nous pencherons sur la mise en place d'un chiffrement de type gestionnaire de mot de passe, évoqué dans la partie III.1. La démonstration finale consistera à faire synchroniser des mots de passe entre 2 appareils et ce avec les 3 moyens possibles.

V) Planning du projet

Diagramme de Gantt du projet:



VII) Introduction Partie expérimentale

Suite à notre étude théorique, nous avons donc commencé à concevoir et programmer notre solution de gestionnaire de mots de passe.

Nous nous sommes tout d'abord réparti les tâches de façon à avancer sur les 3 méthodes de synchronisation de manière uniforme. Notre but au départ était de voir si l'implémentation des 3 méthodes allait être possible donc nous avons cherché à nous familiariser avec l'utilisation des librairies que nous avions sélectionnées pour voir si nous arrivions à les prendre en main facilement.

Nous nous sommes orientés sur le langage Python qui paraissait le plus optimal pour utiliser les 3 méthodes (grâce notamment à des libraires déjà existantes pour utiliser les QrCode, le bluetooth et ICE).

Une fois que nous avions décidé du langage de programmation et que nous savions qu'il était possible d'implémenter les 3 méthodes, nous avons commencé la conception et le développement de la solution.

Démarrage rapide:

Les différentes librairies à installer sont:

- Tkinter : apt-get install python3-tk
- OpenCv : pip install opencv-python
- PyAutoGui : pip install pyautogui
- PyCryptodome : pip3 install pycryptodome==3.4.3
- Aioice : pip install aioice
- QrCode : pip3 install qrcode
- Pillow : pip install pillow
- PyBluez : pip install pybluez==0.42 (la dernière version ne fonctionne pas)

De plus, il y a quelques logiciels à installer pour la partie Bluetooth :

- Microsoft Visual C++ 14.0 standalone: Build Tools for Visual Studio 2017 (x86, x64, ARM, ARM64), téléchargeable via Visual Studio : https://visualstudio.microsoft.com/fr/downloads/

- Si sous Windows 10 : Windows 10 SDK (installé avec Visual Studio par défaut)

- Pour faire la synchronisation de 2 appareils via bluetooth, il est préférable d'appairer les 2 appareils avant de tenter la synchronisation.

VIII) Conception du programme

Le but de la solution était d'avoir une interface commune aux 3 méthodes où se trouve toute la partie de sélection du sens de la synchronisation et de la méthode de synchronisation. Et autour de cette interface, nous aurions différents programmes qui peuvent être appelés via l'interface. Il y aurait un programme par méthode de synchronisation.

Pour travailler efficacement à trois, nous avons créé un dépôt gitlab pour le projet constitué de 5 branches : main, develop, qrcode, bluetooth et ice.

Interface commune:

Le but était premièrement d'avoir une simplicité d'utilisation de l'outil pour les utilisateurs, mais le code devait également être simple à prendre en main pour pouvoir par la suite moduler l'outil afin par exemple de rajouter des modes de partage sans devoir changer tout le code en profondeur. Nous avons donc développé en ce sens.

Ceci nous a donc amené à la création d'une interface graphique qui se lance après que l'utilisateur ait entré son mot de passe maître pour s'identifier. Une première fenêtre se lance pour inviter l'utilisateur à choisir s'il veut partager ses mots de passe à un appareil distant ou bien au contraire se synchroniser en récupérant les mots de passe d'un autre appareil. Une fois cela fait, une deuxième fenêtre apparaît pour cette fois choisir le mode de partage (QRCode, Bluetooth, ICE). Quand l'utilisateur a choisi, le code correspondant à ses deux choix s'exécute (exemple: reçevoir des mots de passe via QRCode).

Pour mettre cela en oeuvre, nous avons créé deux classes énumérées, "Actions" et "Methods", possédant respectivement les valeurs none, sync-this, sync_other et none, qrcode, bluetooth, ice, qui vont nous permettre de sauvegarder les choix de l'utilisateur pour identifier l'action à effectuer.

Ainsi, on crée deux instances de Actions et Methods, initialisées à none, et lorsque l'utilisateur choisit son action "envoyer" ou "reçevoir", on met à jour l'instance de Actions, et on met à jour de même l'instance de Methods quand l'utilisateur a choisit son mode de partage sur la deuxième fenêtre.

A ce stade, nous avons un couple (action,method) nous permettant d'identifier le code à exécuter pour l'échange de mots de passe, il nous suffit alors d'utiliser deux "if" imbriqués afin de faire un switch en fonction des cas, et d'appeler la fonction correspondante, préalablement importée des fichiers QRCode, Bluetooth et ICE.

Procédure à suivre pour intégrer une nouvelle action:

Dans interface.py:

- Ajouter le nom de l'action dans l'Enum Actions à la suite des autres.
- Dans la fonction fenêtre_action(), ajouter le bouton de la nouvelle action de la même manière que le bouton "Envoyer" par exemple. Il va donc falloir associer une fonction à exécuter lors de l'utilisation de ce bouton.
- Créer donc cette fonction de la même manière que Sync_this(fenetre)
- Si l'action à ajouter va différer selon la méthode alors:
  - Ajouter dans le main l'appel à une nouvelle fonction correspondante à l'action voulue.

```
if action == Actions.SYNC_THIS:
    sync_this_device(method)
elif action == Actions.SYNC_OTHER:
    sync_other_device(method)
elif action == Actions.NONE:
    print("You have to chose an action to do")
```

Ceci est nécessaire seulement si l'action va différer selon la méthode. En effet, si on veut par exemple ajouter un bouton "Ajouter un mot de passe" alors on peut exécuter

cette action directement dans la fonction précédemment créée (tiret précédant).
- Il faut donc également créer une nouvelle fonction pour l'action avec en argument la méthode à utiliser de la même manière que sync_this_device(method).

Procédure à suivre pour intégrer une nouvelle méthode de partage:

Dans interface.py:

- Ajouter le nom de la méthode dans l'Enum Methods à la suite des autres.
- Dans la fonction fenêtre_method(), ajouter le bouton de la nouvelle méthode de la même manière que le bouton "QrCode" par exemple. Il va donc falloir associer une fonction à exécuter lors de l'utilisation de ce bouton.
- Créer donc cette fonction de la même manière que QrCode(fenetre2)
- Dans les fonctions gérants une action en fonction d'une méthode comme sync_this_device(method) par exemple, il faut rajouter la vérification avec la nouvelle méthode créée:

```
if method == Methods.NEW_METHOD:
    print("On lance la fonction \"sync_this_device_new_method\" ")
    sync_this_device_new_method()
```

Dans le fichier de la nouvelle méthode:

- Créer les fonctions correspondantes aux actions présentes, par exemple créer la fonction sync_this_device_new_method().

IX) Fonctionnalités du programme

1) Master Password et Chiffrement

Pour ajouter une dimension de sécurité au projet, nous avons mis en place un système de Master Password pour l'authentification (comme sur les gestionnaires de mot de passe classiques) et aussi un système de chiffrement des mots de passe pour sécuriser la synchronisation.

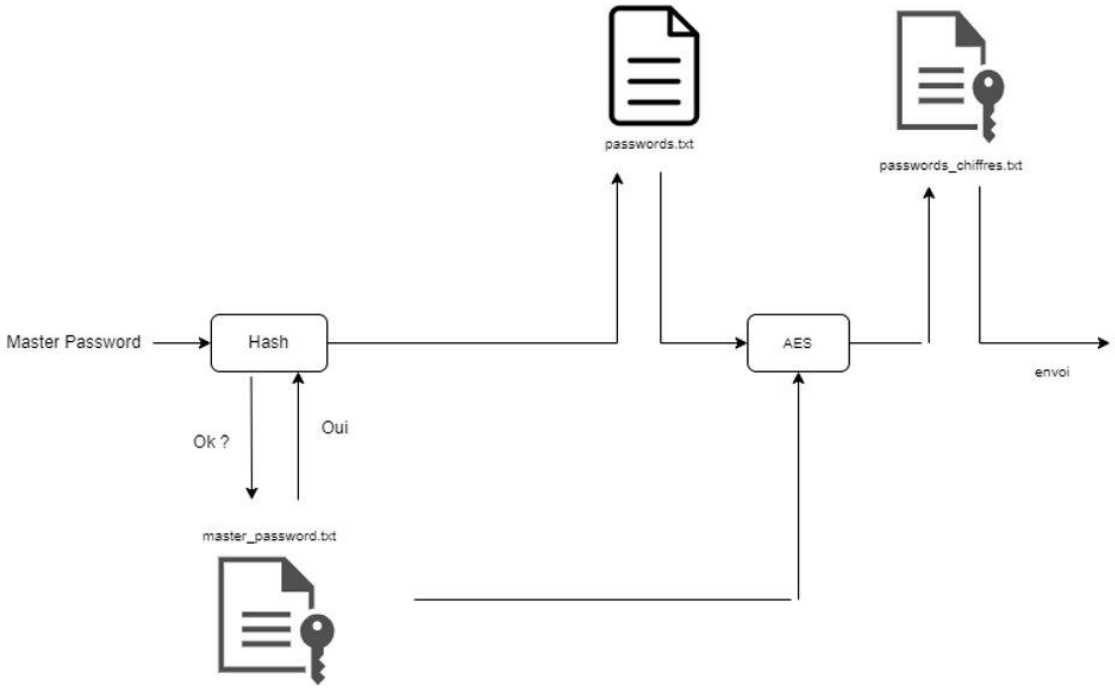
Si un utilisateur veut envoyer ses mots de passe vers un autre appareil pour le synchroniser alors le déroulement de la synchronisation va se passer comme sur le schéma ci-dessous:

Il va d'abord s'identifier en entrant son Master Password, celui-ci va être hashé 100 000 fois et être comparé au hash contenu dans le fichier masterPassword.txt pour savoir si c'est bien le même Master Password. Le hash du Master Password prend environ 1 seconde ce qui rend difficile une attaque par brute force.

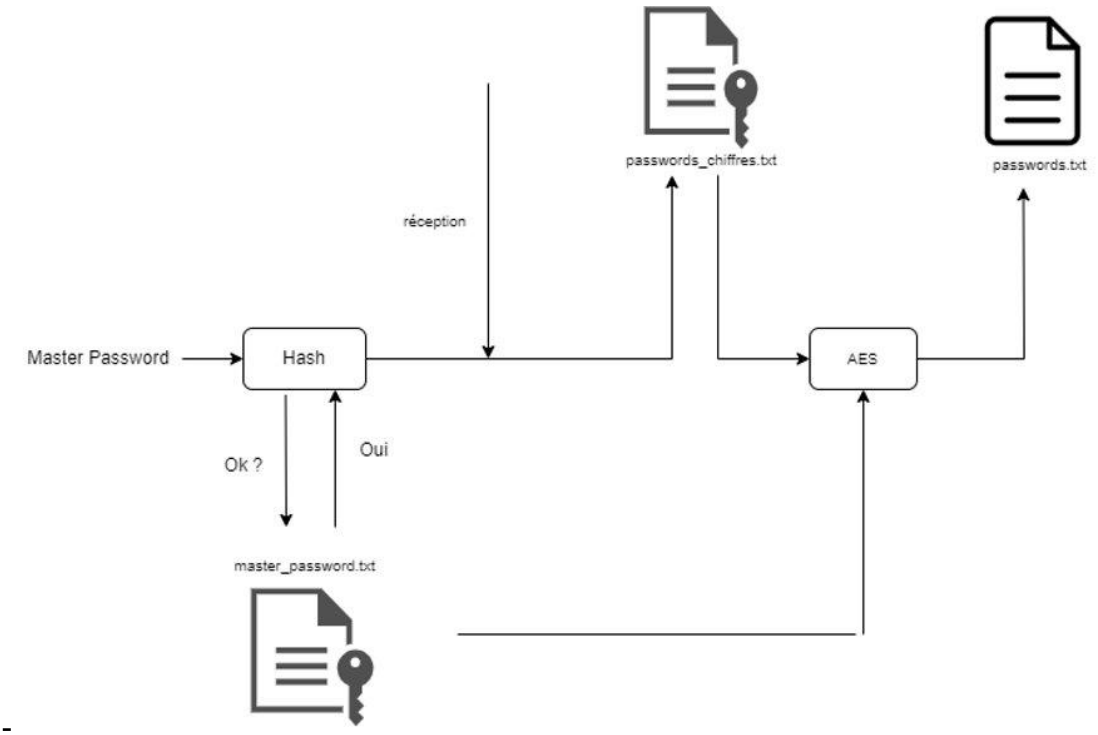
Une fois identifié, peu importe la méthode de synchronisation sélectionnée par l'utilisateur, ses mots de passe qui sont contenus dans passwords.txt vont être chiffrés par AES avec comme clé le hash du Master Password et écrits dans passwordsChiffres.txt. Ainsi, on va pouvoir envoyer les mots de passe chiffrés contenus dans ce fichier pour la synchronisation.

Pour la réception des mots de passe, on récupère via une des méthodes le contenu de passwordChiffres.txt et on peut le déchiffrer en utilisant à nouveau le hash du Master Password. On n'a plus qu'à écrire dans le fichier passwords.txt les mots de passe déchiffrés pour finir la synchronisation.

Voici deux schémas résumant les deux situations d'envoi et de réception :



Déroulement d'un envoi de mots de passe



-

Déroulement d'une réception de mots de passe

2) Synchronisation par Qr Code

Nous disposons d'un fichier QrCode.py où nous avons implémenté les fonctions permettant la synchronisation des mots de passe via la technologie QrCode. Pour ça, nous avons deux fonctions correspondantes aux actions possibles par l'utilisateur: sync_other_device_QrCode() et sync_this_device_QrCode()

- sync_other_device_QrCode():

  Sert pour mettre à jour les mots de passe d'un autre appareil avec les mots de passe présents sur l'appareil actuel.

  Le principe est de générer un QrCode contenant les mots de passe chiffrés à transmettre à un autre appareil, puis de l'afficher à l'écran.
- sync_this_device_QrCode():

  Sert pour mettre à jour ses mots de passe grâce à un QrCode qui a été préalablement généré par un autre appareil avec la fonction précédente.

  Cette fonction va d'abord chercher à ouvrir la caméra de l'appareil. Si cela a réussi, l'utilisateur va pouvoir placer devant sa caméra le QrCode. Il va ensuite pouvoir faire une capture d'écran de ce que voit la caméra en appuyant sur la touche 's' (Attention, pour l'instant la capture s'effectue en haut à gauche de l'écran donc il faut placer la fenêtre de la caméra à cet endroit). On a ainsi une photo du QrCode qui est enregistrée et on va pouvoir lancer une fonction qui va pouvoir détecter et décoder le QrCode sur la photo pour retourner les données du QrCode. Enfin ces données sont utilisées pour mettre à jour le fichier passwords_chiffres.txt. La synchronisation via QrCode est terminée.

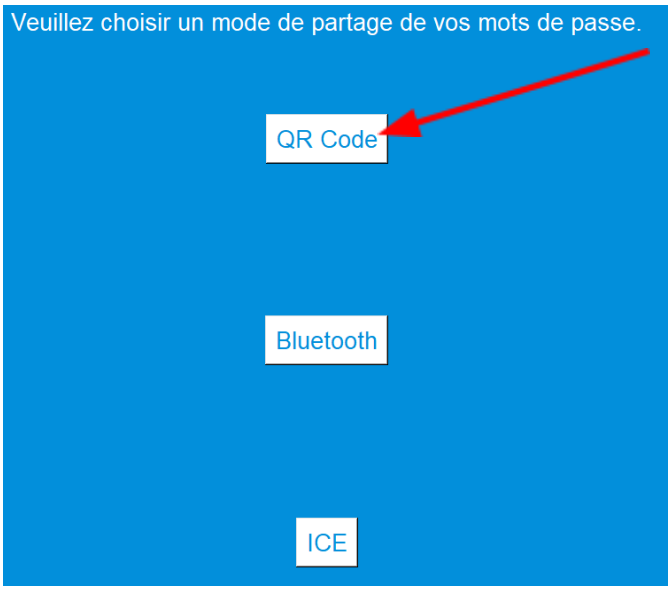
Démo de synchronisation via QrCode:

Contexte: Un utilisateur, Bob, dispose d'un appareil "A" et d'un appareil "B". Il a sur les 2 appareils son gestionnaire de mot de passe. Il ajoute un nouveau mot de passe sur son appareil A. Il veut donc que cet ajout soit également sur son appareil B. Il va alors synchroniser son gestionnaire de l'appareil B grâce à un QrCode généré par l'appareil A.

Sur l'appareil A:

- On choisit d'envoyer notre liste de mot de passe à jour

- On choisit la méthode QR Code pour envoyer nos mots de passe

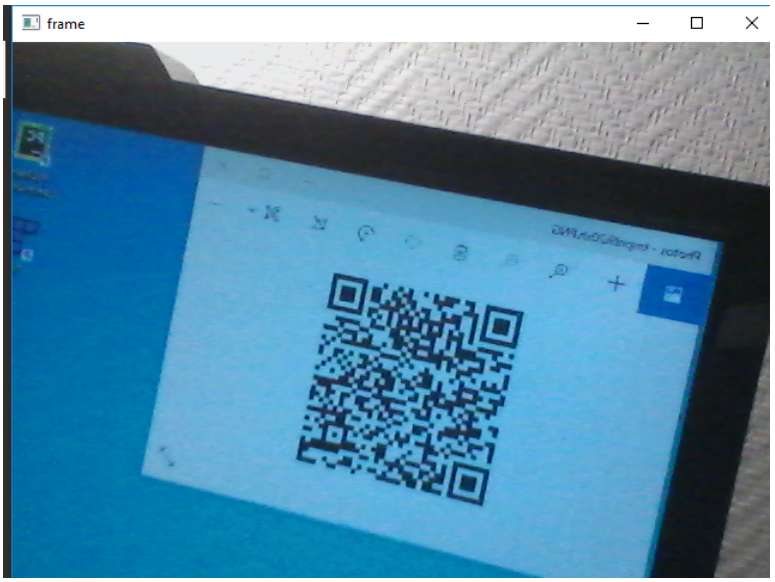


- Un QrCode est alors affiché pour pouvoir être scanné par l'appareil B

Sur l'appareil B:

- Bob choisit cette fois de recevoir une liste de mots de passe

- Il choisit également la méthode Qr Code
- Une fenêtre s'ouvre alors avec la caméra de son pc (pour l'instant le code ne fonctionne que pour un pc sur windows). Il se peut que la caméra ouverte ne soit pas la bonne, l'utilisateur peut alors changer de caméra (en appuyant sur 'c') jusqu'à trouver la bonne. Il place son appareil B devant l'appareil A pour pouvoir prendre en photo le Qr Code de synchronisation:



- Suite à ça, le gestionnaire détecte et décode le Qr Code pour mettre à jour les mots de passe de l'appareil B:

```
detection reussie
synchronisation terminée
```

Remarque: Pour plus de simplicité, Bob aurait pu prendre en photo avec son smartphone le QrCode puis montrer cette photo à son appareil B. Ce qui peut être aussi pratique pour synchroniser 2 appareils éloignés.

3) Synchronisation par Bluetooth

Bluetooth:

Dans le dossier bluetoothFolder se trouve le fichier Python RunBluetooth.py contenant les deux fonctions sync_other_device_Bluetooth() et sync_this_device_Bluetooth().

sync_other_device_Bluetooth() va devoir chercher les périphériques Bluetooth aux alentours pour pouvoir leur envoyer le fichier passwords_chiffres.txt. Le programme donnera une liste de périphériques disponibles et l'utilisateur devra entrer le numéro du périphérique souhaitée. Le programme tentera ensuite la phase de connexion avec celui-ci et attendra que l'utilisateur clique sur Entrée pour envoyer le fichier des mots de passe chiffrés.

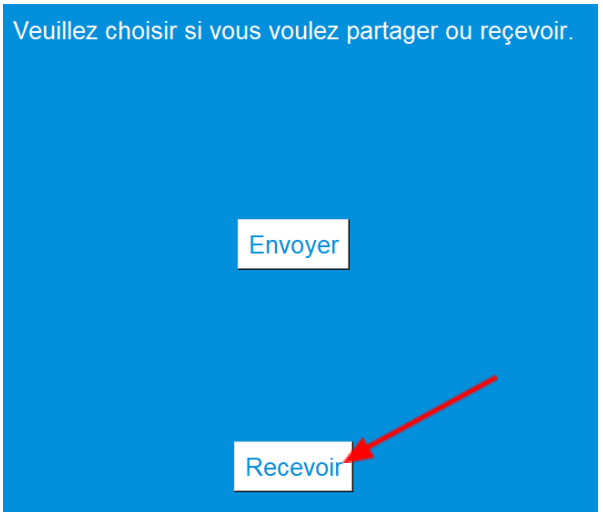
sync_this_device_Bluetooth() va vouloir mettre à jour ses mots de passe et va donc attendre l'initialisation d'une connexion Bluetooth. Une fois la connexion acceptée, le programme recevra les données du serveur et les réécrira dans son fichier de mots de passe chiffrés.

Démo de synchronisation via Bluetooth :

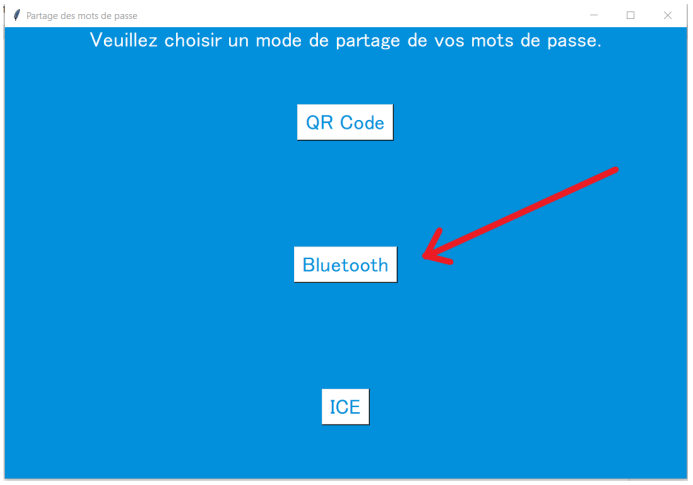
Contexte : Un utilisateur veut synchroniser son fichier de mots de passe présent sur son PC A grâce à celui présent sur son PC B qui est à jour. Les 2 PC sont équipés de Bluetooth.

PC A : Ce PC va lancer *interface.py* et choisir "Recevoir" pour mettre à jour ses mots de passe puis "Bluetooth" pour le choix de la méthode. Le programme écrira ensuite "en attente du serveur…", qui va attendre le lancement du programme sur l'autre PC

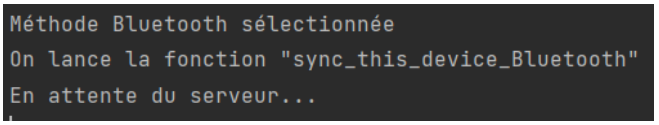
Choix du mode:



Choix de la méthode:

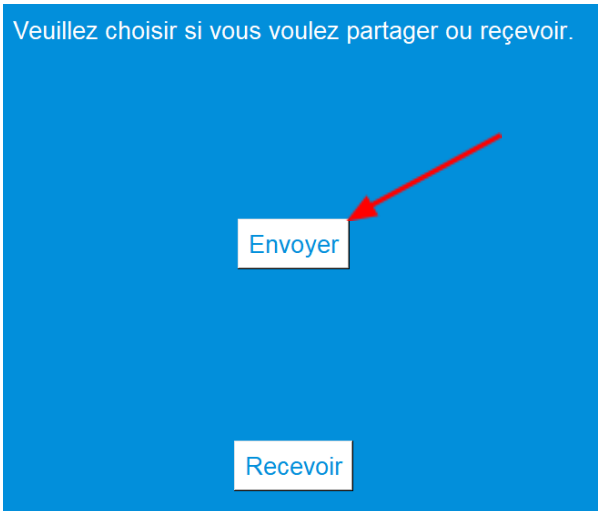


Attente du PC B :

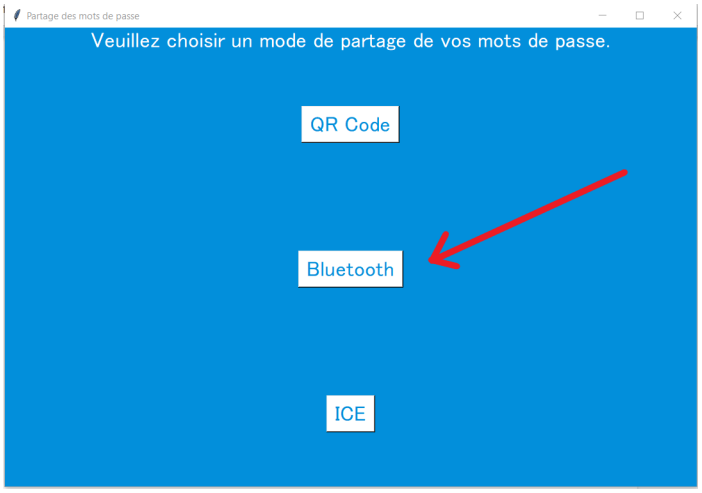


PC B: Ce PC possède la version à jour des mots de passe et va donc l'envoyer au PC A. Pour se faire, on lance également *interface.py*, on choisit ensuite "Envoyer" puis la méthode "Bluetooth". Le programme va alors rechercher les périphériques Bluetooth proches et les lister pour l'utilisateur et il aura à choisir le PC A parmi eux en entrant son index

Choix du mode:



Choix de la méthode:



Choix du périphérique destinataire:

Le programme tente alors de se connecter avec le périphérique Bluetooth sélectionné. En cas de succès, le programme affichera “Connecté” et attendra que l’utilisateur tape sur Entrée pour envoyer le fichier. Le programme indiquera ensuite le succès ou non de l’envoi

Connexion établie et envoi du fichier:

```
Méthode Bluetooth sélectionnée
On lance la fonction "sync_other_device_Bluetooth"
Recherche de périphériques Bluetooth...
addresse 1: D0:27:88:45:DA:D4
nom 1: Wireless Controller
addresse 2: 1C:66:6D:A9:47:DC
nom 2: Wireless Controller
addresse 3: FC:19:99:B1:E5:C4
nom 3: POCO X3 NFC
Entrer l'index du périphérique à qui envoyer les données: 3
Tentative de connexion à FC:19:99:B1:E5:C4 sur le port 0x3...
```

```
Connecté
Cliquer sur Entrée pour envoyer le fichier
Fichier envoyé avec succès!
```

Remarque :
Afin que la connexion soit un succès, il faut que les deux fonctions se fixent un même port sur lequel écouter. L’utilisateur devra changer le port fixé par défaut par notre programme dans le code si celui-ci est déjà utilisé ou indisponible pour l’appareil Bluetooth en question.

4) Synchronisation par ICE

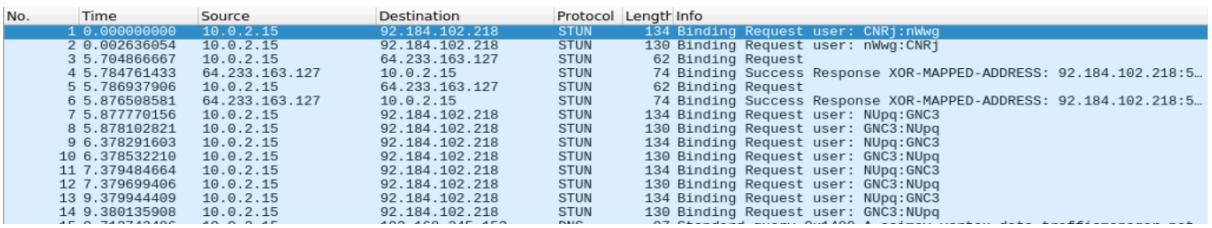
Pour ICE, nous avons utilisé la seule librairie python qui existe pour ce protocole: aioice.

Comme il a été expliqué plus haut, la première étape est de récupérer les adresses (qu'on appellera des candidats) auprès du serveur STUN/TURN. Premièrement, il faut préciser que l'on trouve très peu de serveurs TURN publics, et ceux qui existent ne fonctionnent plus. Il a donc été décidé de se passer du candidat fourni par le serveur TURN et de n'utiliser que celui retourné par le serveur STUN ainsi que l'adresse privée. On a donc choisi un serveur STUN public appartenant à google: "stun1.l.google.com". Puis on l'a interrogé avec des messages STUN, et il nous a bien renvoyé notre adresse publique.

Disposant ainsi de notre adresse privée et de notre adresse publique sur chaque appareil, il fallait ensuite que l'on arrive à créer une connexion entre les deux, en élisant un couple de candidats grâce à ICE. Pour cela, nous avons rentré sur chaque appareil les deux candidats de l'appareil opposé, et nous avons ensuite distingué deux étapes:

- Celle où les adresses privées suffisent à établir la connexion: c'est-à-dire que nous sommes sur le même réseau local, ou que les appareils ne sont pas derrière un NAT. Dans ce cas, ce sont les deux adresses privées qui sont élues, la connexion est établie directement et on peut procéder à l'échange de données.
  Nous avons illustré ce cas en créant deux connexions sur le même appareil et en ouvrant une connexion grâce à ce protocole, puis nous avons procédé à un envoi de données. Cette partie a fonctionné sans problèmes majeurs.

- Celle où les deux appareils sont derrière deux NAT différents : dans ce cas nous avons besoin de l'adresse publique pour pouvoir communiquer avec l'appareil distant. Nous avons tenté de faire la même chose que pour le cas où nous n'avions besoin que des adresses privées, c'est-à-dire de renseigner les candidats mais cette fois sur deux appareils bien distincts et éloignés, puis de tenter d'ouvrir une connexion. On a observé que les candidats élus étaient bien les bons, mais il est apparu que l'ouverture d'une connexion résultait sur le code d'erreur :"ICE negotiation failed". Nous avons essayé en créant deux connexions sur le même ordinateur et en ne donnant que les adresses publiques comme candidat (afin que ce ne soit pas l'adresse privée qui soit élue pour la communication), mais le problème persiste.
  Nous avons effectué une capture wireshark pour voir plus en détail les messages envoyés:



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.2.15	92.184.102.218	STUN	134	Binding Request user: CNRj:nWwg
2	0.002636054	10.0.2.15	92.184.102.218	STUN	130	Binding Request user: nWwg:CNRj
3	5.704866667	10.0.2.15	64.233.163.127	STUN	62	Binding Request
4	5.784761433	64.233.163.127	10.0.2.15	STUN	74	Binding Success Response XOR-MAPPED-ADDRESS: 92.184.102.218:5…
5	5.786937906	10.0.2.15	64.233.163.127	STUN	62	Binding Request
6	5.876508581	64.233.163.127	10.0.2.15	STUN	74	Binding Success Response XOR-MAPPED-ADDRESS: 92.184.102.218:5…
7	5.877770156	10.0.2.15	92.184.102.218	STUN	134	Binding Request user: NUpq:GNC3
8	5.878102821	10.0.2.15	92.184.102.218	STUN	130	Binding Request user: GNC3:NUpq
9	6.378291603	10.0.2.15	92.184.102.218	STUN	134	Binding Request user: NUpq:GNC3
10	6.378532210	10.0.2.15	92.184.102.218	STUN	130	Binding Request user: GNC3:NUpq
11	7.379484664	10.0.2.15	92.184.102.218	STUN	134	Binding Request user: NUpq:GNC3
12	7.379699406	10.0.2.15	92.184.102.218	STUN	130	Binding Request user: GNC3:NUpq
13	9.379944409	10.0.2.15	92.184.102.218	STUN	134	Binding Request user: NUpq:GNC3
14	9.380135908	10.0.2.15	92.184.102.218	STUN	130	Binding Request user: GNC3:NUpq

On peut voir que chaque connexion effectue bien une requête de connexion au NAT, mais on ne reçoit jamais de réponse.

On arrive donc actuellement à partager des mots de passe en utilisant le protocole ICE mais seulement en local, et la barrière du NAT n'est pas franchie lorsque l'on veut passer sur une communication distante. Une option est peut-être que le protocole ICE n'est pas connu par le NAT qui ne peut donc pas effectuer le relayage.

X) Pistes d'amélioration

Avec plus de temps, nous aurions pu ajouter certaines fonctionnalités qui peuvent être utiles à cette solution:

- Synchronisation via le réseau local:
  Une possibilité qui avait été évoquée lors de l'étude théorique était que pour 2 appareils sur un même réseau local, la solution la plus simple pour les synchroniser était de les faire communiquer via UDP ou TCP directement. Le but du projet étant de proposer une synchronisation pour 2 appareils qui ne sont pas sur le même réseau local, nous n'avons pas implémenter cette synchronisation. Cependant, si ce gestionnaire devait être finalisé, cette synchronisation serait très importante.
- Faire une version Android/IOS:
  Pour l'instant, nous avons programmé un code uniquement compatible avec Windows mais l'idéal serait de faire une version compatible pour les smartphones. En effet, pour un utilisateur de gestionnaire de mots de passe, il est pratique que son smartphone dispose aussi du gestionnaire de mot de passe et qu'il puisse le synchroniser avec son pc par exemple.
- Meilleure interface:
  Pour notre programme, nous avons besoin d'avoir une interface utilisateur pour qu'il puisse dans un premier temps s'identifier et ensuite choisir ce qu'il veut faire. Pour l'instant nous avons fait une interface fonctionnelle pour tester notre programme mais il faut évidemment la refaire plus proprement. Sachant aussi que pour le moment, le master password est demandé sur le prompt et lors de son écriture, il est affiché en clair.
  De la même manière, l'utilisation des différentes méthodes pour un utilisateur lambda manque de clarté. Par exemple, pour la méthode Qr Code, il faut appuyer sur la touche 's' pour prendre un screenshot mais ce n'est pas mentionné sur l'interface. Il faudrait des indications pour que l'utilisateur sache quoi faire exactement pour chaque méthode.

XI) Conclusion

Notre projet avait pour but de créer une solution de gestionnaire de mots de passe permettant une synchronisation entre différents appareils de manière décentralisée, en particulier en évitant l'utilisation de serveurs pour stocker les mots de passe.

Après avoir étudié un maximum de moyens de communication entre deux appareils, les avoir comparés, dégagé leurs contraintes et leur respect de la règle de décentralisation, nous avons pu discerner trois options dont l'ensemble nous permettait une synchronisation simple, à la fois locale et distante : l'utilisation des QR Codes, du Bluetooth et du protocole ICE.

Afin de rendre notre solution simple d'utilisation, nous avons premièrement créé une interface destinée à l'utilisateur et lui permettant de choisir en cliquant sur des boutons s' il veut envoyer ou reçevoir des mots de passe, et par quelle méthode. Nous avons ensuite mis en place un chiffrement des mots de passe afin de n'envoyer que des mots de passe chiffrés sur le réseau. Enfin, nous avons développé les trois méthodes de partage évoquées précédemment. Le QR Code nous permet de créer un QR Code contenant les données des mots de passe d'un côté, et de scanner ce QR Code ainsi généré de l'autre afin de récupérer les mots de passe. La méthode Bluetooth permet de créer une connexion Bluetooth entre deux appareils et d'envoyer les mots de passe puis de les récupérer dans un fichier.

Le protocole ICE couplé à STUN nous a permis de pouvoir récupérer facilement notre adresse IP publique ainsi que de partager nos mots de passe localement avec notre adresse ip privée, mais il n'a pas permis un partage à distance, la barrière du NAT n'arrivant pas à être franchie.

Enfin, nous avons évoqué quelques axes d'amélioration de notre solution, comme l'adaptation à une version sur téléphone, qui ont tout autant besoin d'un gestionnaire de mots de passe qu'un ordinateur de nos jours.