I. Étude théorique

Ce projet vise à concevoir un gestionnaire de mots de passe (GMP) qui nécessite moins de capacité de calcul et de stockage pour un serveur centralisé. Ceci est dans l'optique d'avoir un GMP qui soit gratuit pour les utilisateurs et à moindre coût pour l'hébergeur afin de démocratiser l'utilisation de ces derniers.

Un tel GMP doit donc utiliser des protocoles légers et ne se basant pas sur une machine centrale qui orchestre l'ensemble du système. Le GMP doit assurer ces fonctions tout en étant simple d'utilisation et très sécurisé. L'étude théorique a donc permit de définir les fonctionnalités desservant ces contraintes :

- Stockage décentralisé et sécurisé de fichiers
- Transmission sécurisé des fichiers entre plusieurs machines
- Synchronisation et versionning des fichiers
- Gestion simple des appareils utilisateurs

Des contraintes moins "métiers" sont en ambition du projet, à savoir :

- Robustesse de la solution (multi-plateforme)
- Adaptations aux diverses architectures réseaux (diverses méthodes NAT)

Le projet ne s'intéressera qu'aux problématiques autours de ces fonctionnalités et n'ira pas étudier en profondeur les fonctionnalités élémentaires d'un GMP comme le chiffrement local des fichiers. Il s'agit donc avant tout de réaliser une preuve de concept fonctionnelle sur les fonctionnalités et caractéristiques visées.

Les première solutions qui peuvent apparaître sont l'utilisation d'un système décentralisé comme le Peer-to-peer. À cela viennent s'ajouter des questions sur la synchronicité des communications entre les appareils d'un utilisateur. Tout cela sera donc étudié dans l'état de l'art ainsi que les solutions de GMP déjà existantes.

II. État de l'art

Les gestionnaires de mot de passe

Actuellement, diverses solutions de GMP existent déjà. Nous avons donc fait le tour de celles-ci afin de voir si des solutions répondant à nos contraintes existaient déjà. Nous avons pu observer qu'il en existe différents types :

Local *Exemple : Keepass*

Tous les identifiants/mots de passe entrés dans le gestionnaire sont chiffrés avec un mot de passe maître dans un fichier. Ce fichier est stocké sur le disque dur. L'accès à ce fichier est donc protégé par un secret connu uniquement par l'utilisateur.
Toutefois il faut noter que mis à part une copie manuelle du fichier d'un PC à un autre, l'utilisation des mots de passe est restreint au PC où le fichier chiffré est stocké. Il existe des extensions

permettant de synchroniser le fichier sur un drive comme Dropbox, or cela complique beaucoup l'utilisation de l'outil.

Online *Exemple : Dashlane*

Ici, l'ensemble des mots de passe est centralisé dans une base de données sur un serveur public. Le fonctionnement est similaire à celui décrit précédemment, les mots de passe sont chiffrés localement sur une machine avec une clé maître et génère un fichier chiffré ensuite stocké dans un serveur.

Sans états *Exemple : Lesspass*

Une autre solution consiste à calculer le mot de passe à chaque utilisation afin de ne pas le stocker. Une fonction déterministe prend en paramètre un mot de passe maître, l'URL du site, l'identifiant, l'adresse mail etc… puis calcule le mot de passe en local, sur la machine. L'utilisateur doit donc se souvenir de tous ces paramètres. Il est impossible de stocker quoi que ce soit d'autre, MDP ou notes sécurisées.

Blockchain *Exemple : You.*

Certaines solutions proposent d'utiliser des blockchains pour stocker le fichier de MDP. Cette méthode est intéressante car elle utilise un système complètement décentralisé et accessible à tous. Cependant, le fichier de MDP est alors publiquement accessible sur la blockchain et dépend de la Blockchain utilisée. À ce jour, les Blockchain sont trop instables car les Cryptomonnaies sont basées sur cette technologie. Utiliser une Blockchain reviendrait donc à faire fructifier une Cryptomonnaie, ce qui n'est pas notre but.

Bilan

Pour résumer, le modèle Online apporte toutes les fonctionnalités que l'on recherche si l'on omet les contraintes sur la liberté d'utilisation et le coût des infrastructures. Le modèle de Blockchain apporte justement des solutions à ces contraintes mais nous ne pouvons l'utiliser. Nous allons donc nous inspirer de ces modèles qui nous intéressent pour avoir une base avant de trouver les solutions qui nous conviennent.

Les architectures Peer-to-peer

Principe

Le P2P est un modèle de réseau différent du modèle client-serveur. Dans ce modèle toutes les machines du réseau sont au même "niveau". Chaque machine est à la fois client et serveur.

Centralisé

Sur ce modèle, il y a tout de même un serveur central qui héberge un index des ressources du réseau P2P. Il est donc rapide pour un pair de chercher dans cet index quelle ressource est sur quel pair.

Décentralisé et structuré

On peut aussi décentraliser un index des ressources entre les nœuds du réseau. Les pairs ont juste à partager l'information sur la topologie et l'emplacement des ressources. Par contre, un tel modèle devient vite complexe et a une structure assez rigide qui amène des résultats de recherche erronés.

Décentralisé et non-structuré

On peut aussi ne pas avoir d'index du tout sur le réseau P2P. Il n'y a donc aucun contrôle sur la topologie ou l'emplacement des ressources. Dans ce cas les pairs doivent faire des requêtes en Broadcast à tous les pairs dans un certains "rayons". Ce modèle est très robuste mais plus il y a de pairs, moins il est efficace.

Les communication synchrone et asynchrone

Dans la majorité des réseaux pair à pair, lorsqu'un échange de données se fait entre les pairs, on a une connexion synchrone. C'est à dire que les deux pairs qui communiquent sont tous les deux allumés pendant la communication et donc que le transfert d'information est limité aux appareils connectés en même temps au réseau.

Or, vu l'usage qui sera fait de notre GMP, on souhaite transférer les modifications du fichier sans toutefois que les autres appareils soient allumés. Pour cela nous allons donc mettre en place un système asynchrone où le serveur ne stockera pas indéfiniment les données mais de manière temporaire comme le fonctionnement d'une boîte au lettre.

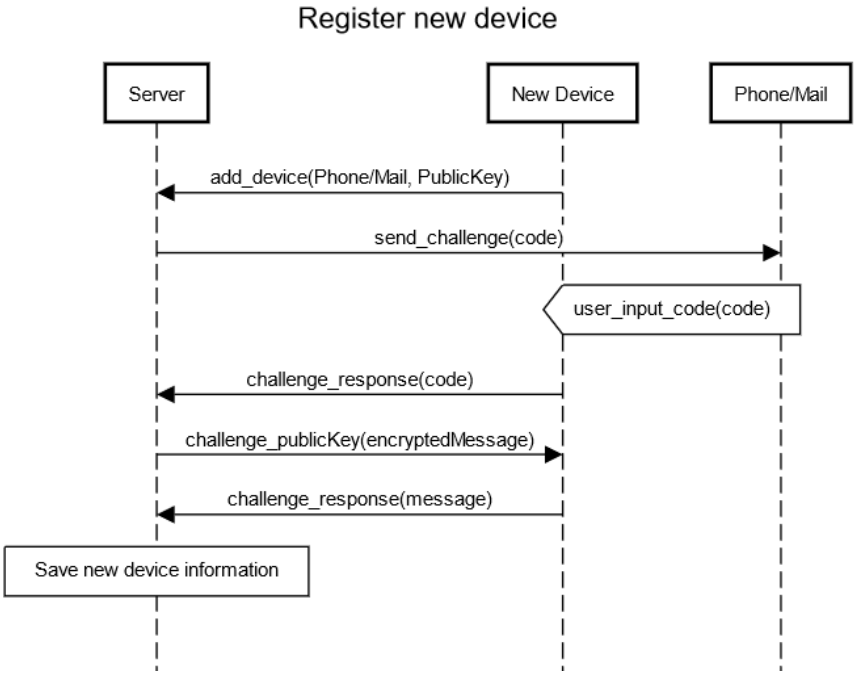
III. Étude expérimentale

Dans l'étude expérimentale nous cherchons à définir de manière plus précise quels sont les objectifs du projets et quels moyens nous allons mettre en œuvre pour les tenir.

Tout d'abord, les MDP des utilisateurs doivent être stockés en local sur le PC et chiffrés avec un algorithme symétrique fort (proposition de CHACHA). La clé de chiffrement devra être dérivée depuis un mot de passe maître que l'utilisateur doit retenir pour accéder aux MDP. Le fichier de MDP est aussi chiffré une deuxième fois avec une autre clé de chiffrement sur laquelle nous reviendrons juste après.

Vient alors la problématique de synchronisation de ce fichier entre les différents périphériques de l'utilisateur. Concernant le versionning du fichier de MDP une solution basée sur Git (seulement les fonctionnalités push, pull, commit) est pour le moment envisagée.

L'utilisateur a donc besoin que ces périphériques soient enregistrés quelque part. Cela se fait donc sur le serveur et voici un diagramme montrant ce processus :



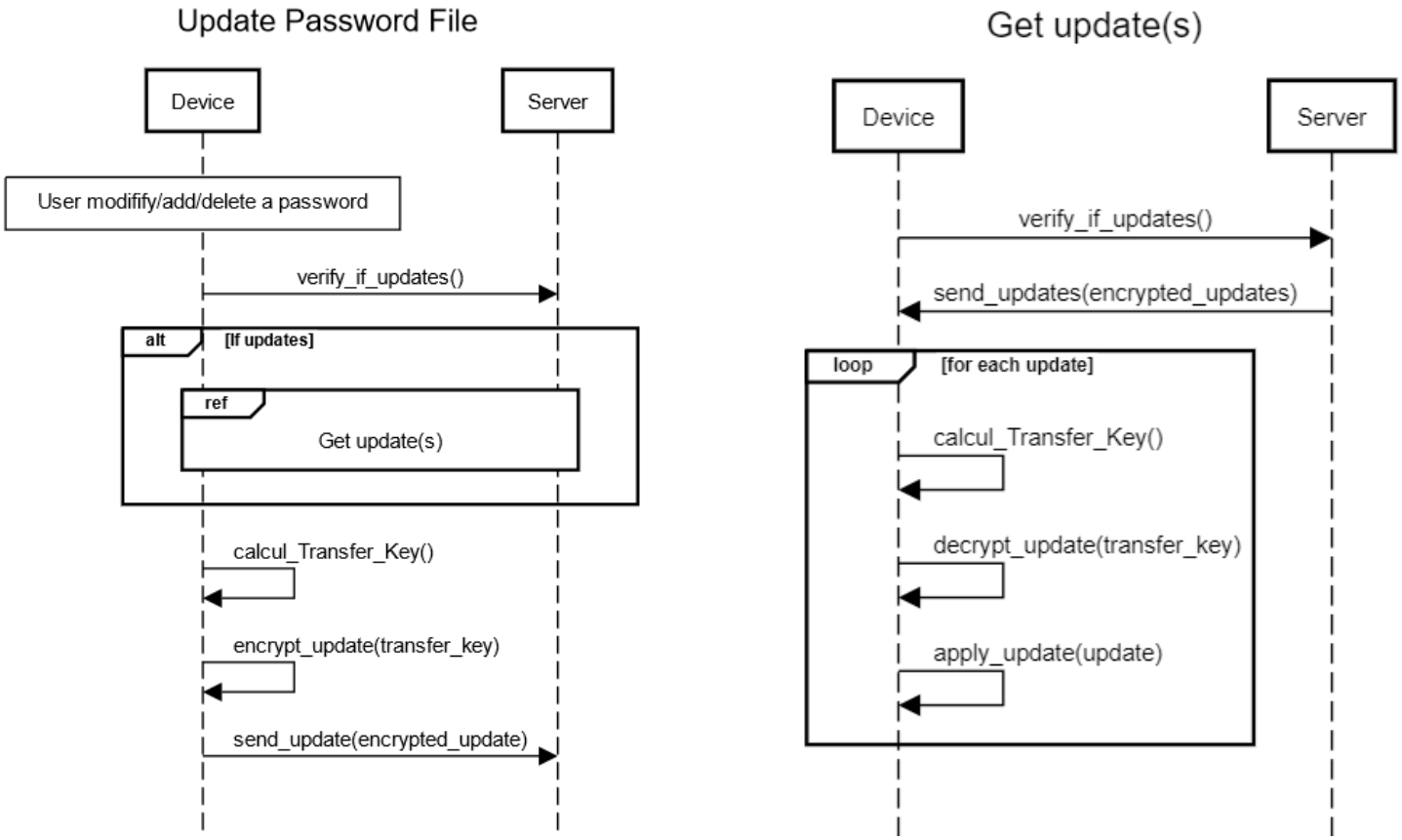
On peut donc voir que l'utilisateur avec son nouvel appareil va s'identifier avec une adresse email ou un numéro de téléphone. Cet identifiant sera alors challengé par le serveur pour que l'utilisateur puisse prouver son identité et l'associer au nouvel appareil. Une fois ce challenge réussi, le serveur va envoyer un message chiffré avec la clé publique de l'appareil et ce dernier doit montrer qu'il peut déchiffrer avec sa clé privée. Une fois la clé publique associée à l'appareil, le serveur peut enregistrer cette clé pour l'identifier dans les communications à venir.

Ensuite, pour le transfert du fichier, nous avions deux options, soit les machines partagent une clé de transfert identique pour toutes, soit elles établissent deux à deux une clé de session pour ne discuter qu'avec une seule machine en face. Nous avons choisi d'utiliser une clé de transfert commune et ce choix sera expliqué dans le rapport final.

Ainsi, les machines vont tout d'abord générer une clé de session symétrique qui sera partagée entre toutes les machines par un échange Diffie-Hellman (DHE). Cette clé de session est utilisée pour initialiser un Double Ratchet qui assurera une protection des communications précédentes et futures. Ici, un Ratchet est une fonction de dérivation de clé qui empêche de trouver la valeur à partir de laquelle il génère la nouvelle clé. C'est pour cela qu'on parle d'un Ratchet (roue à rochet), car on ne peut que "tourner" dans un sens. Le Double Ratchet est alors un algorithme qui, avec un premier Ratchet basé sur la clé de session échangée par DHE, permet de synchroniser nos machines. Ensuite, l'issu de ce premier Ratchet sert dans un deuxième Ratchet qui gère le renouvellement continu et la maintenance de la clé de transfert à courte durée de vie. Le Double Ratchet va donc générer à chaque fois une nouvelle clé de transfert servant à chiffrer (proposition de AES-256) les mises à jour du fichier de MDP à transférer aux autres machines. Comme le Double Ratchet est identique sur chaque machine, la clé de transfert l'est aussi.

On utilise donc le même principe de chiffrement et de transfert que ce soit à l'initialisation du fichier de MDP ou pour une mise à jour de ce dernier.
Dans les deux cas, la donnée doit être envoyée sur le serveur et récupérée par les autres appareils pour assurer un fonctionnement asynchrone. C'est-à-dire que le serveur gardera la donnée tant que le destinataire ne l'a pas récupéré. Cela ne pose pas de problème de confidentialité car le serveur ne va stocker que temporairement les données chiffrées sans avoir la possibilité de les déchiffrer.

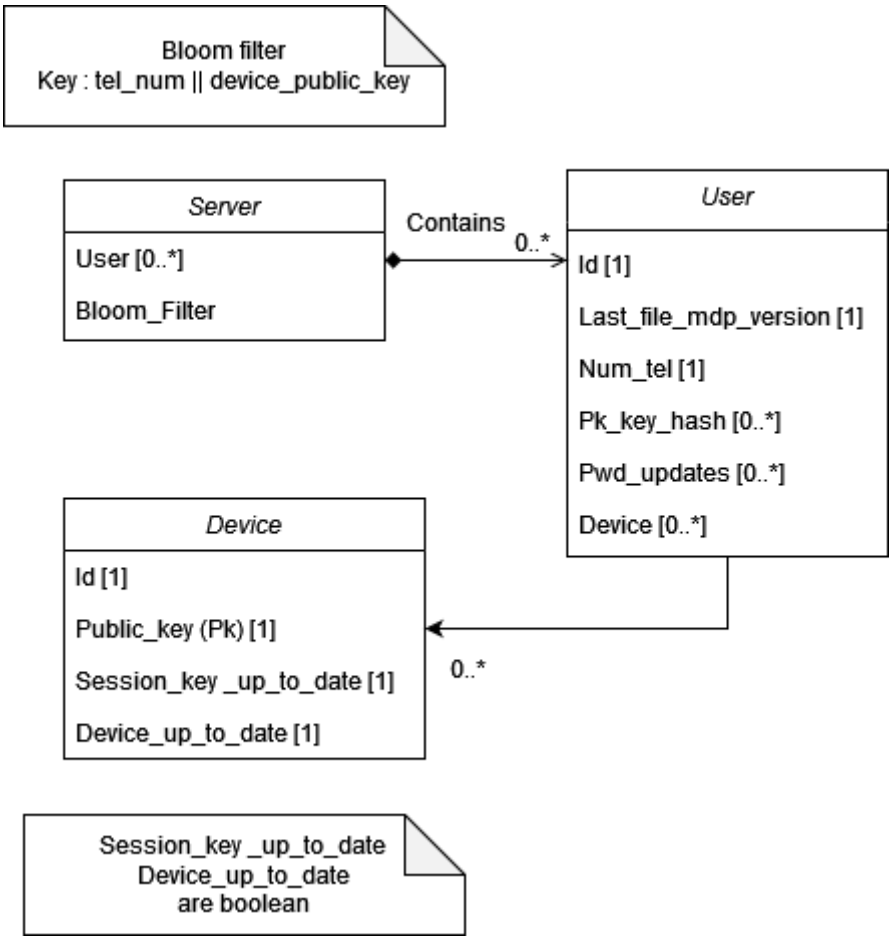
Voici donc deux diagrammes montrant comment, en deux temps, le transfert de mise à jour peut se faire :



Tout d'abord, le périphérique souhaitant pousser une mise à jour doit vérifier qu'il n'y en a pas déjà à récupérer. Après cela, il peut calculer une nouvelle clé de transfert grâce au Double Ratchet puis chiffrer et envoyer la mise à jour.

Ensuite, de manière asynchrone, un autre périphérique pourra demander au serveur s'il y a des mises à jour à récupérer et les télécharger. Pour chaque mise à jour il devra donc calculer une nouvelle clé de transfert pour déchiffrer la mise à jour reçue et l'appliquer de son côté.

Pour aller plus loin concernant le serveur, un certain nombre d'informations sur les utilisateurs et leurs appareils doit être stocké sans perdre de vue l'objectif d'optimiser la consommation d'espace mémoire et de puissance de calcul. Voici ce que contient le serveur sous la forme d'un diagramme de classe :



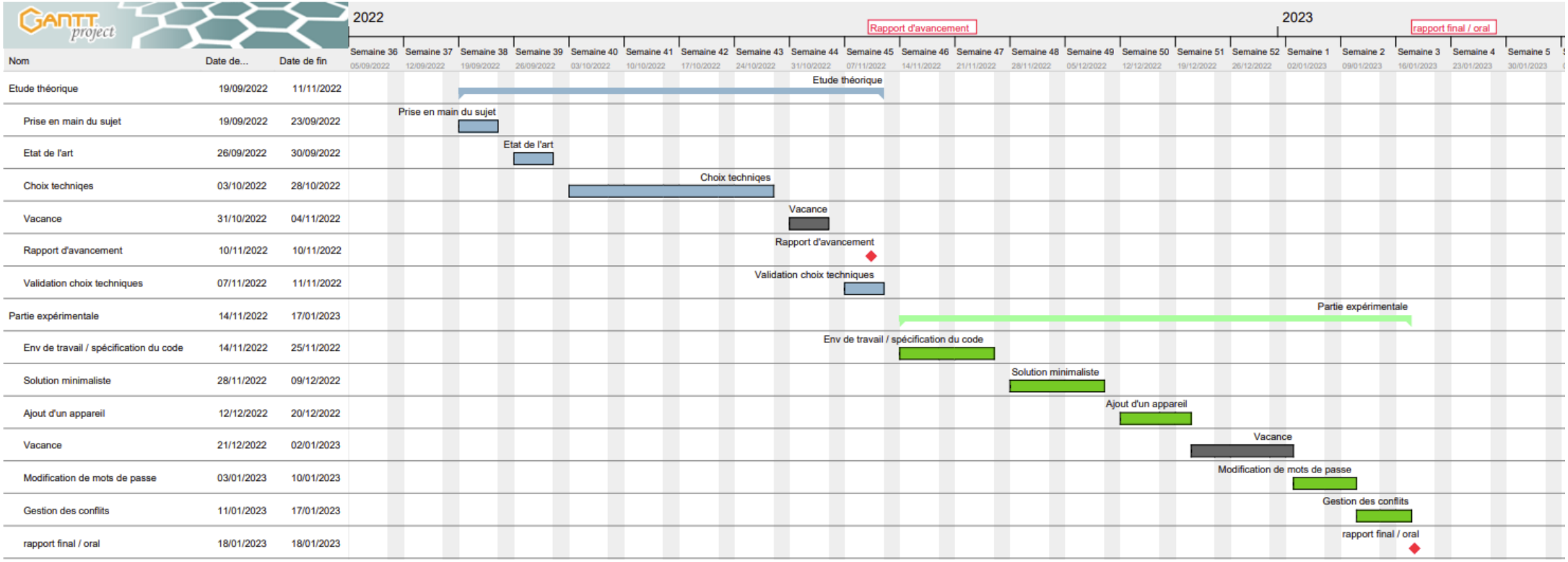
Ainsi, le serveur possède une liste d'utilisateurs et un filtre de bloom. Ce filtre permet de vérifier qu'une clé publique correspond à l'utilisateur qui envoie une requête au serveur et par conséquent limiter les attaques par déni de service non ciblées.

Un utilisateur possède un identifiant unique, le numéro de version la plus récente du fichier de mot de passe commun à tous les appareils, les différentes modifications du fichier de mot de passe (supprimées après un timeout ou lorsque tous les appareils sont à jour). Il est nécessaire pour le serveur de connaître les clés publiques de tous les périphériques associés à l'utilisateur ainsi que le numéro de téléphone de l'utilisateur.

Enfin, les informations associées aux appareils correspondent à un numéro d'identifiant, la clé publique utilisée lors d'échanges DHE et 2 flags. Le flag session_key_up_to_date permet au serveur de savoir quels sont les périphériques pour lesquels il est nécessaire de communiquer la clé de session. Le flag app_up_to_date indique que l'appareil est à jour sur le fichier de MDP.

IV. Planning du projet

Diagramme de Gantt



Nous effectuons des réunions hebdomadaires avec notre chargé de projet pour faire un point sur l'avancement, s'assurer que l'on parte dans le bon sens et revoir les travaux effectués la semaine passée.