Faculté des Sciences
Aix*Marseille Université

Rapport de stage

Master 2 – Fiabilité, Sécurité et Intégration Logicielle
Parcours Fiabilité et Sécurité Informatique

Conception & développement système de backup chiffré et incrémental en C/C++

Arksens Cyber Security

Auteur :
Ludovic Lubeigt

Tuteur Entreprise :
Gaëtan van Diemen

Enseignant :
Jean-Luc Massat

Année Universitaire :
2014 – 2015

Résumé

Ce document expose le travail réalisé lors du stage de Master 2 Fiabilité et Sécurité Informatique à l'Université d'Aix-Marseille. Celui-ci s'est déroulé à Arksens Ltd. à l'île Maurice entre le 1er avril et le 11 septembre 2015.

Le sujet de stage était de créer la nouvelle génération de système de sauvegarde de l'entreprise. Celui-ci devait être un système sécurisé, robuste et multi-plate-forme. Ainsi les données seraient chiffrées sur la machine client avant d'être envoyées vers un serveur. Il ne serait alors pas possible pour une quelconque personne ayant accès au serveur de lire ces données.

Dans un premier temps un travail de recherche et de réflexion a été mené pour aider à la conception de cette application.

Une fois celle-ci réalisée, le développement a pu commencer. À la fin de ce stage, le développement était bien avancé même si non terminé faute de temps.

Mots-clés : sauvegarde incrémentale, sécurisé, *C++*, base de données, multi-plate-forme, conception, développement, cryptographie, chiffrement, multi-thread, Open Source, client/serveur

Remerciements

Je tiens tout d'abord à remercier l'équipe pédagogique de la faculté des sciences de l'université d'Aix-Marseille qui m'a permis d'avoir les connaissances et les aptitudes nécessaires au bon déroulement de ce stage.

Je remercie également la société Arksens pour m'avoir permis de faire mon stage chez eux. Tout particulièrement, je remercie David Terranova et Gaëtan van Diemen qui m'ont accueilli à l'île Maurice et suivi au jour le jour durant ces six mois, mais également Michaël Colaone et le reste de l'équipe pour l'accueil et la bonne ambiance au sein de l'entreprise.

Enfin je remercie toutes les personnes qui m'ont aidé dans le cadre du stage, dans l'écriture de ce rapport ou qui m'ont tout simplement fait découvrir l'île Maurice.

Je tiens tout particulièrement à remercier Sir Daniel Brands of House Brands, the first of His Name, King of the Dutch, King of the flatlands and the first Men, Breaker of Cars and Father of the Bikers.

Table des matières

I Rapport **6**

1 Introduction **7**

2 Présentation de l'entreprise **8**
2.1 Présence dans le monde 8
2.2 Produits, solutions et offres 9
2.2.1 Les produits et solutions de sécurité 9
2.2.2 Les offres de cloud 9
2.3 L'idéologie de l'entreprise 10
2.3.1 La politique des centres de données 10
2.3.2 La culture Open Source 10
2.4 Île Maurice 11

3 Sujet de stage **13**
3.1 Contexte 13
3.2 Le sujet 14

4 Déroulement du stage **15**
4.1 Travail de recherche 16
4.1.1 Documentation 16
4.1.2 Réflexion 18
4.1.3 Prototypage 19
4.2 Conception 19
4.2.1 Un programme modulaire 19
4.2.2 Interaction entre modules 20
4.2.3 Base de données 22
4.2.4 Stockage des fichiers 28
4.2.5 Communication client - serveur 28
4.2.6 Algorithme de cryptographie 29
4.3 Développement 30
4.3.1 Outils de développement 30
4.3.2 Travail réalisé 30
4.3.3 Tests 33

5 Conclusion **34**

II Annexes **38**

A Base de données relationnelle **39**
A.1 User . . . 39
A.2 Machine . . . 39
A.3 File . . . 40
A.4 Chunk . . . 40

B Protocole de communication **41**
B.1 Definition . . . 41
B.2 Transport and Authentication . . . 41
B.3 Messages . . . 41
B.4 Types . . . 42
B.4.1 0 - Client Config . . . 42
B.4.2 1 - Index && 7 - Index Update . . . 44
B.4.3 2 - Request . . . 46
B.4.4 3 - ResponseData . . . 46
B.4.5 4 - ReponseFileInfo . . . 47
B.4.6 5 - Ping . . . 48
B.4.7 6 - Pong . . . 48
B.4.8 8 - Close . . . 48

Première partie

Rapport

1 Introduction

Dans le cadre du Master professionnel *Fiabilité, Sécurité et Intégration logicielle*, parcours *Fiabilité et Sécurité Informatique* à l'*Université d'Aix-Marseille*, un stage de fin d'étude doit être effectué en entreprise pour valider les acquis et entrer dans le monde du travail.

J'ai réalisé ce stage entre le 1er avril et le 11 septembre 2015, soit une période de cinq mois et demi, dans l'entreprise *Adhara Cyber Security*, renommée *Arksens* au 1er juillet.

Ce stage fut donc l'occasion pour moi de mettre en pratique mes connaissances, acquises tout au long de mon parcours universitaire, dans un environnement professionnel. Ma mission durant ce stage a été de concevoir et dévélopper un système de sauvegarde incrémentale et sécurisé, offrant un chiffrement en local des données des utilisateurs. J'ai procédé à la rédaction du plan de développement puis à l'implémentation de celui-ci. J'ai ainsi participé au processus de création, jusqu'à un stade avancé, de ce qui peut être qualifié de gros projet.

Ce rapport présente le déroulement du stage ainsi que le travail accompli au sein de l'entreprise durant ces six mois.

2 Présentation de l'entreprise

Créée en 2013 sous le nom d'*Adhara Cyber Security* avant d'être renommée *Arksens* au 1er juillet 2015, l'entreprise dans laquelle j'ai effectué mon stage est spécialisée en sécurité informatique. Elle se développe sur trois continents grâce à une approche novatrice et répondant aux besoins des entreprises et administrations de toutes tailles.

Le changement de nom a fait suite à une évolution des clients puisque l'entreprise, bien que principalement prestataire de service pour des PME s'est ouverte aux entreprises de taille plus importante. Revoyant leur stratégie de commercialisation et de communication, l'entreprise se devait donc de changer de nom.

2.1 Présence dans le monde

Aujourd'hui *Arksens* est donc présent dans trois pays chacun sur un continent différent (voir figure 2.1) offrant ainsi aux utilisateurs un service de proximité :

— Abu Dhabi aux Émirats Arabes Unis pour les activités au Moyen Orient ;
— Pamplemousses à l'île Maurice pour les activités africaines ;
— Paris en France pour les activités européennes.

FIGURE 2.1 – Lieux où trouver Arksens.

2.2 Produits, solutions et offres

2.2.1 Les produits et solutions de sécurité

Arksens propose aux entreprises 6 produits et solutions autour de la sécurité informatique. Une rapide description de ces produits se trouve table 2.1 :

Produit	Description
mail BY ARKSENS	Protéger l'information transitant par les mails. Hébergement de serveur email sécurisé dans un cloud dédié.
whisper BY ARKSENS	Communications voix et images sécurisées et anonymes.
backup BY ARKSENS	Système de chiffrement et de sauvegarde des données dans un cloud dédié.
Gateway BY ARKSENS	Sécuriser et rendre anonyme la navigation sur Internet.
endpoint BY ARKSENS	Sécuriser les postes utilisateurs en local. C'est une ligne de défense pour palier à la diffusion d'éventuels logiciels malveillants (virus, chevaux de Troie, vers, logiciels espions,...) sur les ordinateurs et les serveurs.
nomad BY ARKSENS	Sécuriser les données personnelles sur les appareils mobiles.

TABLE 2.1 – Produits et solutions par *Arksens*

L'ensemble des produits et solutions proposés par l'entreprise sont décrits de manière plus complète sur leur site web [15].

2.2.2 Les offres de cloud

L'entreprise propose également des offres autour du Cloud [1]

— Offre 1 - Pure cloud

1. Exploitation de la puissance de calcul ou de stockage de serveurs distants par l'intermédiaire d'un réseau, généralement l'Internet.

- 100 % des services sont hébergés ;
- La gestion est assurée par *Arksens* ;
- Des fonctionnalités supplémentaires sont proposées telles que l'hébergement de fichiers ou de logiciels.

— Offre 2 - Hybrid cloud
- Un hébergement local avec une "box" chifrée et une réplication sur le cloud *Arksens* ;
- La gestion est assurée par *Arksens* ;
- Idéal pour les clients souhaitant une réplication dans leurs locaux.

— Offre 3 - Private cloud
- Le cloud est présent uniquement chez le client dans une salle de serveurs ou dans un centre de données ;
- La gestion est assurée par *Arksens* ;
- Idéal pour les clients souhaitant stocker l'ensemble des services et des données localement (comme les banques, la défense, l'audit...).

Avec trois offres de cloud comportant l'ensemble des solutions de sécurité citées précédemment, *Arksens* peut répondre à l'ensemble des besoins de ses clients actuels.

2.3 L'idéologie de l'entreprise

2.3.1 La politique des centres de données

Pour chaque service, *Arksens* fournit le matériel (centres de données), le produit et le support qui va avec. Pour être en mesure de fournir les meilleurs services, elle possède une politique de sélection des centres de données qu'elle utilise. Ci-dessous une liste représentant les différents aspects les plus importants que doivent respecter les centres de données :

- Uniquement des Tier IV (Haute disponibilité [2] : 99,995 %) ;
- Toutes les données sont chiffrées sur les serveurs dédiés ;
- Les centres de données doivent se situés dans des pays libres respectant le *Patriot Act* [5].

De ce fait l'entreprise a actuellement des centres de données en France chez OVH [11] qui lui confère de plus une solution contre les attaques DDoS [3] et en Suisse chez AlpineDC [2] dont la législation est une des plus restrictive en matière de protection de données personnelles.

2.3.2 La culture Open Source

L'Open Source est une tradition dans l'entreprise. L'ensemble des produits respectent les critères établis par l'Open Source Initiative [10], à savoir :

- Possibilité de redistribution ;
- Accès au code source ;
- Création de travaux dérivés.

Ce choix permet notamment d'accentuer la confiance avec les clients et d'améliorer le niveau de sécurité des produits grâce aux avantages suivants :

- Logiciel indépendant : aucune porte dérobée (*backdoor*) [4] ne peut être introduite par une organisation externe puisque les produits sont développés par *Arksens* pour *Arksens* ;

2. Mesure de performance. C'est le temps durant lequel le système est opérationnel par rapport à sa durée totale d'exploitation
3. Attaque par déni de service. Le principe est d'envoyer une multitude de requête depuis plusieurs sources différentes pour rendre un service indisponible.
4. Fonctionnalité inconnue de l'utilisateur légitime et qui donne un accès secret au logiciel

— Communauté : l'accès facile aux produits permet de créer une communauté d'utilisateurs capable sur le long terme de proposer un support sur les produits développés, des améliorations, mais aussi permet de déceler des bogues ;
— Code source accessible : tout le monde a accès au code source, rien n'est caché, la crédibilité et la qualité des produits est ainsi mise en avant.

2.4 Île Maurice

L'île Maurice abrite les locaux accueillant le centre de recherche et développement de l'entreprise. Les locaux se trouvent au Business Park de Beau Plan (figure 2.2) à Pamplemousses, ville située au nord-ouest de l'île, à proximité de Port Louis. C'est donc là que j'ai effectué mon stage.

FIGURE 2.2 – Beau Plan Business Park

L'équipe a beaucoup évoluée entre mon arrivée et mon départ puisque seulement trois personnes en plus du PDG, Michaël Colaone, étaient présentes au 1er avril, date de début du stage : David Terranova, directeur des opérations, Gaëtan van Diemen, chef de projet ainsi que Daniel Brands, développeur web arrivé quelques jours auparavant.

L'équipe s'est agrandie par deux fois. D'abord à la mi-avril avec l'arrivée de deux autres stagiaires, Didier Mannone et Yves Colin de Verdière, puis au 1er mai avec l'arrivée d'Aymeric Tabourin, ingénieur sécurité.

À mon départ, une restructuration était en cours et plusieurs changements allaient être apportés au niveau de l'équipe en place avec notamment le départ de certains collaborateurs (licenciement, démission ou tout simplement fin de stage). Ceci a précipité la fin de mon stage d'une quinzaine de jour puisque j'ai arrêté celui-ci au 11 septembre bien qu'il était initialement prévu de finir le 25 septembre.

Ces changements montrent que la vie d'une startup peut rapidement évoluer, que ce soit dans un sens ou dans l'autre. En l'espace de six mois seulement une période de recrutement s'en était donc suivi d'une restructuration importante impliquant le départ de plusieurs employés.

Durant mon stage, la présence de collaborateurs néerlandais et mauriciens a permis la création d'un

environnement international, bien que fortement francophone. Afin de pouvoir communiquer avec l'ensemble des personnes de l'équipe, l'utilisation de l'anglais au quotidien était donc une nécessité.

L'intégration dans l'équipe s'est donc faite assez facilement et rapidement. C'est ainsi dans une ambiance généralement bonne que s'est déroulé ce stage de fin d'étude, bien que quelque peu compliquée sur la fin dûe à la restructuration de l'entreprise.

3 Sujet de stage

3.1 Contexte

L'entreprise étant encore jeune et de petite taille, plusieurs des services existants étaient basés sur des produits Open Source et avaient été intégrés au *manager* (voir figure 3.1).

Ce *manager* est un environnement web développé et maintenu par *Arksens*. Il permet aux utilisateurs de gérer les services auxquels ils ont souscrit à partir d'un unique endroit centralisé et ainsi faciliter l'utilisation de ces derniers.

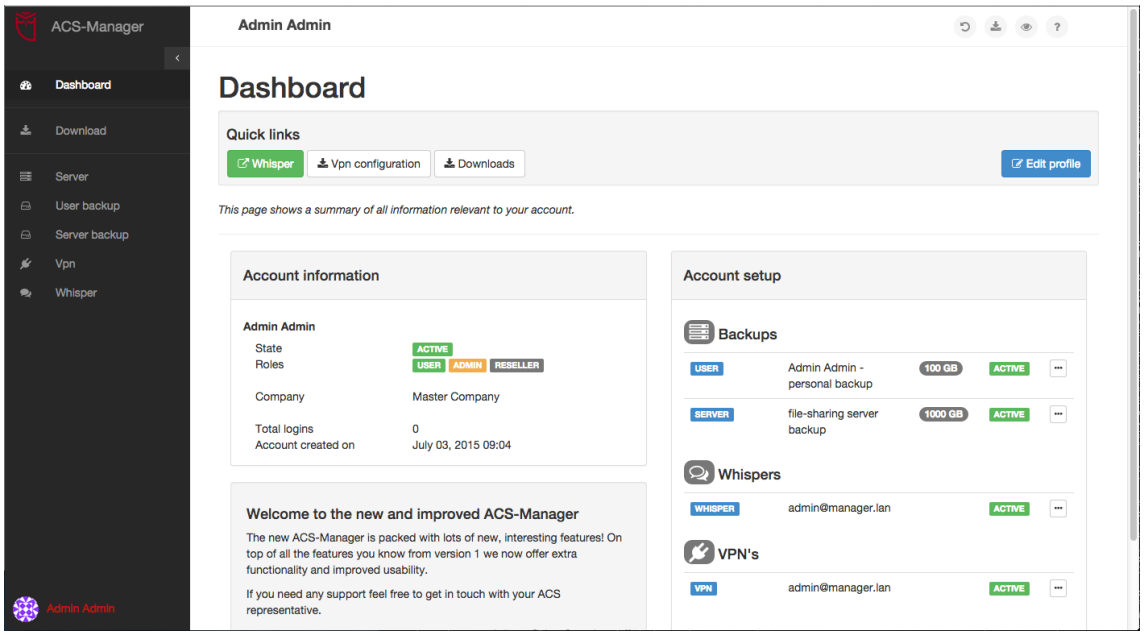


FIGURE 3.1 – Page d'accueil du *manager*

L'objectif du recrutement important était pour l'entreprise de petit à petit développer ses propres solutions afin de proposer aux utilisateurs des produits adaptés et répondant à leurs besoins tout en minimisant l'utilisation de logiciels tiers.

De plus les systèmes de backup ne sont souvent pas considérés par les entreprises avant qu'une perte de données, plus ou moins importante, ne survienne. Conséquences financières, temps passé à reconstituer le savoir évaporé ou encore l'impact sur la réputation de l'entreprise ne sont que des exemples de ce qui peut arriver dans une telle situation. Pour prévenir ce problème un système de backup est non seulement nécessaire mais presque obligatoire.

Pour cela, *Arksens* voulait offrir à ses clients un système de sauvegarde fiable, sécurisé et optimisé qui pouvait être exécuté sur les ordinateurs les moins performants. De plus la prise en compte de connexions

Internet pouvant être parfois très lentes était une obligation. En particulier, une partie du marché visé par *Arksens* se trouve en Afrique où l'accès à un Internet rapide n'est pas toujours aussi facile que ce qu'il peut y avoir en Europe.

C'est donc dans ce cadre que je suis arrivé et qu'il m'a été demandé de créer la prochaine génération de backup pour *Arksens*.

3.2 Le sujet

Ma mission pour ce stage était donc de faire la conception et le développement d'un système de sauvegarde incrémentale. Les données sauvegardées sont chiffrées en local, sur la machine client, avant d'être envoyées sur des serveurs distants. Il me fallait donc développer un client/serveur multi-plate-forme, robuste et sécurisé répondant à la problématique suivante : combiner le chiffrement local avec la sauvegarde incrémentale.

De plus le programme devra être *multi-threadé* pour pouvoir optimiser la sauvegarde des données. Entre autres, cela permettra de chiffrer des blocs constituants les fichiers tout en envoyant au serveur ceux déjà chiffrés.

Par ailleurs, comme ce service allait être proposé aux entreprises, il devait être pris en compte qu'un administrateur puisse gérer les différentes sauvegardes ou être informé si un problème survenait sur le poste d'un employé lors de l'utilisation du logiciel.

Enfin, un système de quota devait être mis en place. Ce système permettrait aux entreprises de demander un plus ou moins grand quota selon la quantité de données à sauvegarder. En effet celle-ci pouvant varier selon la taille ou la fonction de l'entreprise, un quota permettrait une certaine flexibilité.

Cette application permettrait donc, au premier lancement, de sauvegarder l'ensemble d'un ou plusieurs dossiers. Par la suite, seules les modifications apportées aux fichiers seraient sauvegardées.

À mon arrivée, David et Gaëtan avaient déjà émis des idées sur la manière de réaliser ce système de sauvegarde, idées que nous verrons plus loin dans ce rapport. C'est à partir de celles-ci que j'ai pu démarrer mon travail, qui sera alors divisé en trois grandes phases :

— Phase de documentation et de réflexion sur la problématique ;
— Conception logicielle ;
— Développement.

La première phase de documentation et de réflexion consistait à rechercher différentes solutions déjà existantes et potentiellement exploitables avant de procéder à toute phase de conception.

La conception s'est faite en prenant en compte les contraintes existantes et en se basant sur un travail de réflexion déjà effectué par David et Gaëtan avant mon arrivée.

Enfin, en ce qui concerne le développement, une des contraintes que j'avais, était d'utiliser le *C++* et de développer de manière orientée objet tout en faisant en sorte que le code puisse être robuste, lisible, testable, et donc facilement maintenable. De plus les librairies et *framework* utilisés devaient être sous licence libre afin d'être en droit de publier l'ouvrage sous licence libre. Une dernière contrainte, mais pas des moindres, était de développer un logiciel pouvant tourner sur les plate-formes répandues : *Mac OSX*, *Windows* et *GNU/Linux*.

Pour mener cette mission à bien, il avait été décidé qu'une réunion hebdomadaire ait lieu avec David et Gaëtan. Celle-ci pour qu'ils puissent suivre l'avancement du projet, voir le travail effectué la semaine précédente, m'aider à résoudre les éventuels problèmes et planifier la semaine à venir. C'est en tout cas dans cette optique que nous avions commencé. Dans les faits, le retour des clients de l'entreprise lors de mises à jour des produits ou des réunions avec de potentiels futurs clients pouvaient prendre la priorité et ainsi retarder ou annuler les réunions hebdomadaires.

4 | Déroulement du stage

Avant de commencer le projet en lui-même, David et Gaëtan m'ont fait part de leur réflexion sur le possible fonctionnement du système de sauvegarde.

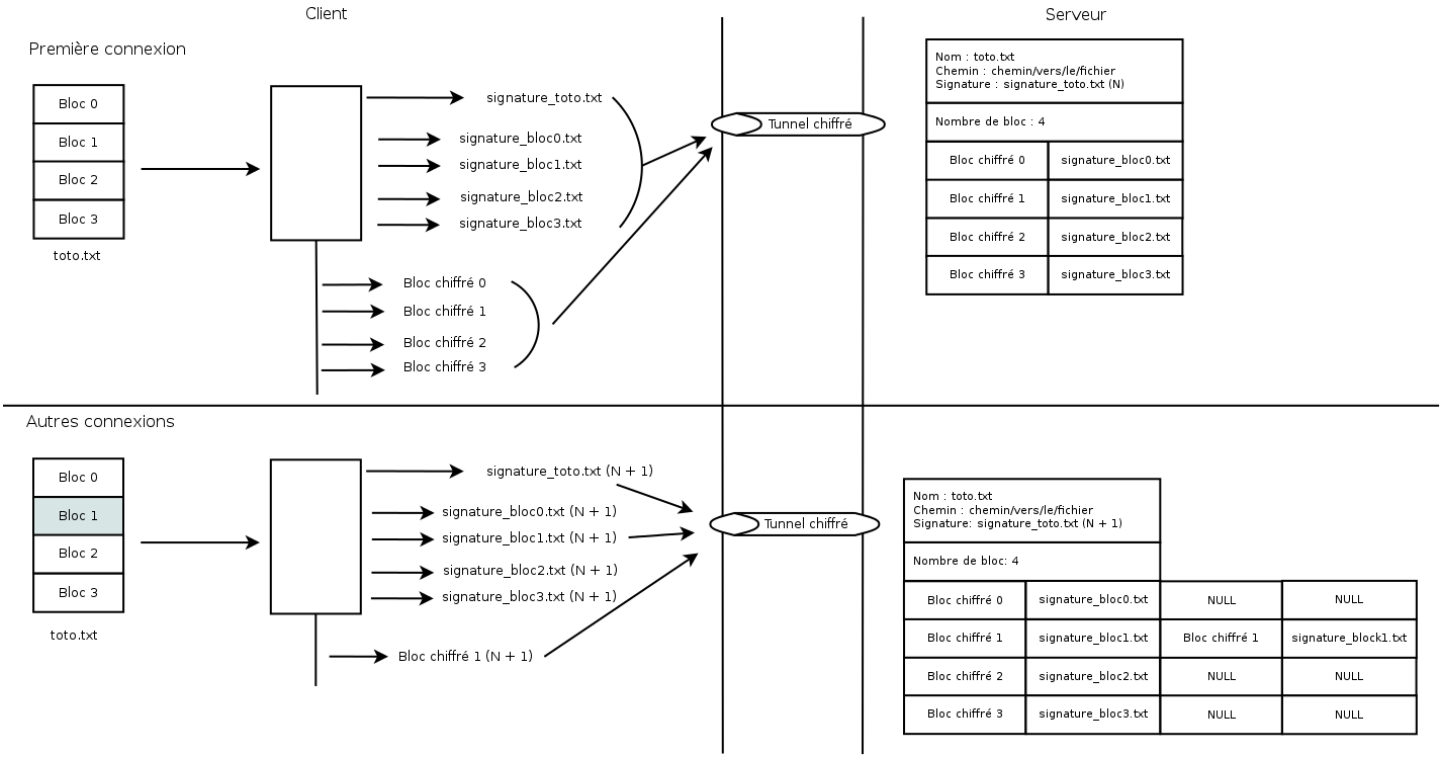


FIGURE 4.1 – Schéma initial du fonctionnement du backup

Le système possède deux comportements différents lors de la sauvegarde d'un fichier.

D'abord à la première connexion, où une sauvegarde intégrale est nécessaire, puis lors des connexions suivantes où seulement les modifications doivent être envoyées.

Dans le premier cas, le fichier est découpé en blocs de taille égale avant d'être chiffré puis envoyé accompagné des signatures calculées pour chaque bloc ainsi que pour le fichier en lui même. Les signatures permettant de vérifier l'intégrité des fichiers.

Le second cas est un peu plus complexe et est décrit comme suit :

— Calculer *signature_toto.txt* $(N+1)$;
— Télécharger *signature_toto.txt* (N) ;
— Comparer les deux signatures : si elles sont identiques, le fichier n'a pas été modifié, on passe donc à la vérification du fichier suivant. Sinon, on continue ;
— Découper *toto.txt* en bloc ;

— Calculer la signature de chaque bloc $(N+1)$;
— Télécharger la signature de chaque bloc (N) ;
— Comparer chaque signatures entre elles ;
— Pour chaque signature différente, envoyer sur le serveur le bloc chiffré correspondant accompagné de sa signature $(N+1)$.

Dans le cas de la figure 4.1, le bloc 1 étant modifié, sa signature, à $N+1$, ne correspond pas à la signature précédemment calculée (N). Le bloc est donc chiffré avant d'être envoyé sur le serveur avec sa signature *signature_ bloc1.txt*.

L'idée bien qu'intéressante ne peut être appliquée que dans un cas très particulier de fonctionnement, lorsque les modifications n'affectent pas la taille d'un bloc et que l'ordre des blocs est resté inchangé.

Pour tout les autres cas (ajout, suppression, déplacement de tout ou partie d'un bloc, etc.), l'algorithme ne pourrait pas fonctionner. Pour palier à cela, j'ai donc dû imaginer un algorithme répondant à la problématique et fonctionnant dans tous les cas imaginables.

4.1 Travail de recherche

4.1.1 Documentation

Mon travail de recherche s'est fait à partir d'une liste de mots-clés qui m'a été donné afin que l'on parle tous le même langage. Entre autres, certaines définitions liées au backup et au chiffrement :

— Backup incrémental [1] ;
— Backup différentiel [2] ;
— Transchiffrement [3].

Par ailleurs, il pouvait être intéressant de s'inspirer des logiciels de backup ou de synchronisation déjà existant :

— Syncthing [14] ;
— rsync [17].

4.1.1.1 Syncthing

Syncthing est un logiciel permettant la synchronisation des données entre plusieurs appareils au travers d'une communication sécurisée via TLS. Les données n'étant jamais stockées sur un serveur tiers, uniquement les différents ordinateurs utilisés y ont donc accès.

L'aspect intéressant de *Syncthing* est pour nous le protocole de communication, qui a été créé à l'occasion : Block Exchange Protocol (BEP) [16]. Ce protocole sous Creative Commons [3], est donc une bonne source d'inspiration pour faire la liaison entre notre client et le serveur.

4.1.1.2 rsync

rsync est un programme de transfert de fichiers pour les systèmes Unix. Le cœur du programme est son algorithme, schématisé figure 4.2 : « *rsync algorithm* » [18].

1. Sauvegarde de fichiers dont le principe est d'envoyer uniquement les fichiers modifiés depuis la dernière sauvegarde effectuée
2. Sauvegarde de fichiers dont le principe est d'envoyer uniquement les fichiers modifiés depuis la dernière sauvegarde complète effectuée
3. Technique consistant, lors d'un changement de clé de chiffrement, à déchiffrer l'ensemble des données avec la précédente clé puis à les re-chiffrer avec la nouvelle

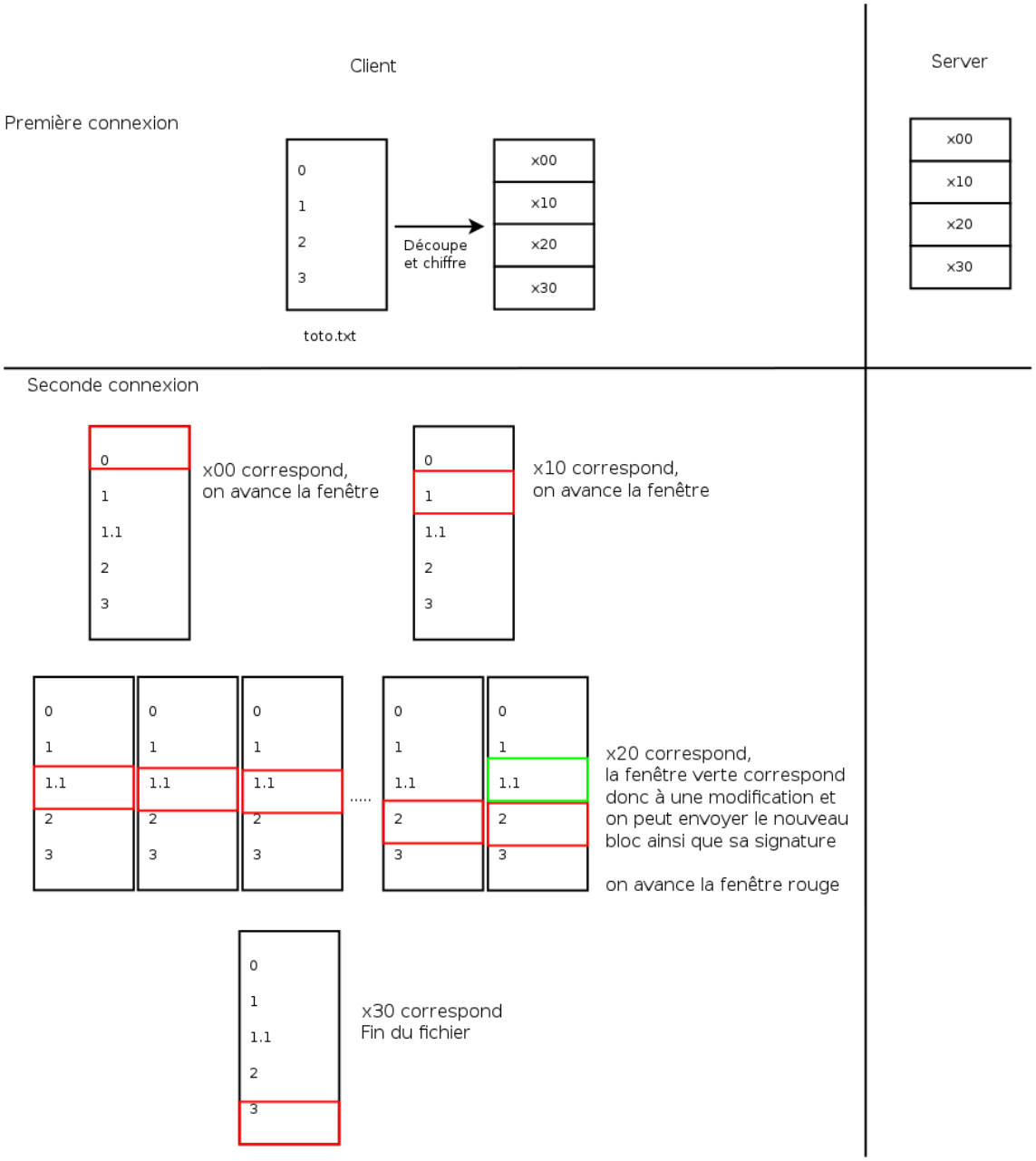


FIGURE 4.2 – Schéma montrant brièvement l'algorithme *rsync*.

Celui-ci va découper un fichier en plusieurs blocs de taille identique avant de calculer, pour chacun, un `checksum` ainsi qu'un `hash`.

Pour n'envoyer que les parties modifiées d'un ficher, l'algorithme va créer une fenêtre de la taille des blocs. Celle-ci va parcourir l'intégralité du fichier octet par octet.

Pour chaque position de la fenêtre, le `checksum` — rapide à calculer mais dont le risque de collision est élevé — est calculé et comparé au `checksum` déjà calculé au précédent backup de chaque bloc composant le fichier. Si un *checksum* correspond, le `hash` — plus lent à calculer mais ayant un risque de collision beaucoup plus faible, selon l'algorithme utilisé — du bloc correspondant est alors comparé. Si celui-ci est identique, la fenêtre correspond à un bloc non modifié du fichier.

Ainsi, tout ce qui précède la fenêtre, et jusqu'au précédent bloc trouvé, correspondant à une partie du fichier modifiée (ajout/modification/suppression) et doit donc être synchronisé.

C'est en grande partie cet algorithme qui m'a permis d'en trouver un répondant à ma problématique. Problématique à la fois similaire au problème de *rsync* puisqu'il fallait comparer deux fichiers distants pour n'envoyer que les changements mais également différent dans la mesure où il n'est pas possible dans mon cas de maintenir une liste de bloc de même taille au delà de la première sauvegarde.

4.1.2 Réflexion

Une fois cette phase de documentation réalisée, il était temps de trouver un algorithme permettant d'identifier les parties d'un fichier ayant été modifié depuis le précédent backup. Pour ce faire, et comme indiqué dans la partie documentation, je me suis basé sur l'algorithme *rsync* pour petit à petit créer un algorithme qui permettait de répondre à mon problème.

4.1.2.1 Chiffrement homomorphe

Dans un premier temps, j'ai étudié la question de l'homomorphisme. Ce type de chiffrement est capable de réaliser des opérations sur des fichiers chiffrés et de retourner le résultat chiffré. Ceci permet de faire des opérations sans avoir à connaître le contenu d'un fichier.

- **Fonctionnement :**
  Dans notre cas, il nous aurait permis de concatener sur le serveur l'ensemble des morceaux d'un fichier avant de le re-découper en bloc de taille égale puis de calculer pour chacun le `checksum` ainsi que le `hash`. Ceci aurait permis d'utiliser directement l'algorithme *rsync* pour identifier les modifications faites sur un fichier ;
- **Avantage :**
  Consommation de ressource et temps de calcul réduit sur la machine client ;
- **Inconvénient :**
  En utilisant le chiffrement homomorphe, le plus gros du calcul se fait sur le serveur entre deux backup, et ceux pour chaque fichier de chaque utilisateur. Ce type de chiffrement est donc trop demandeur de ressources du côté du serveur pour que le programme fonctionne correctement ;
- **Problème :**
  Le chiffrement homomorphe est un domaine de recherche très prometteur mais pas encore assez avancé pour pouvoir être utilisé en production. De plus le temps de calcul est extrêmement élevé. Cette solution ne peut donc pas être utilisée aujourd'hui mais pourra peut-être un jour être envisagée dans une prochaine version du backup.

4.1.2.2 rsync revisité

Étant donné que les fichiers doivent être chiffrés et déchiffrés uniquement sur la machine hôte (le client), nous ne pouvons effectuer d'opérations sur ceux du serveur. Il fallait trouver un algorithme qui permette d'analyser un fichier en un temps fini. J'ai donc décidé de baser mon travail sur le principe de fenêtre glissante utilisée par *rsync*. Ne pouvant pas avoir des morceaux de fichier de taille identique, il a fallu modifier l'algorithme pour prendre ceci en compte. Ainsi avoir une fenêtre dynamique semblait être l'option à prendre.

- **Fonctionnement :**
  L'algorithme utilise une fenêtre dynamique, parcourant l'intégralité du fichier et adaptant sa taille en fonction des longueurs des morceaux de fichier que nous avons ;
- **Avantage :**
  Fait côté client, il n'y a pas une sur-utilisation des ressources serveurs qui est alors utilisé uniquement pour stocker les données une fois chiffrées ;

— **Inconvénient :**
Temps de calcul potentiellement long, dépendant de la taille des fichiers à analyser ainsi que des ressources disponibles sur la machine client.

C'est donc sur cette base que j'ai commencé l'étape suivante. Étape consistant à écrire un prototype permettant de vérifier la faisabilité, notamment en ce qui concerne le temps d'exécution du programme.

4.1.3 Prototypage

La vérification, en pratique, de la théorie était nécessaire avant de pouvoir penser à la conception du produit. Pour cela, le passage par une étape de prototypage était inévitable.

Dans ce cadre, j'ai écrit l'algorithme en *C++* afin de vérifier le temps d'exécution de celui-ci. Plusieurs essais ont été faits afin de réduire toujours plus la durée d'exécution de l'algorithme. C'est donc principalement de l'optimisation, à la fois au niveau algorithmique et au niveau de l'écriture du code en lui-même, que j'ai effectuée au cours de cette étape.

Plusieurs versions ont donc vu le jour au cours de cette étape afin d'obtenir un algorithme qui puisse être utilisable pour le système de sauvegarde. Le prototype écrit ayant permis de valider le fonctionnement de l'algorithme conçu, j'ai pu passer à l'étape suivante et commencer ainsi la conception.

4.2 Conception

4.2.1 Un programme modulaire

La conception a commencé avec le découpage du projet en différents modules. Avant de penser à la partie serveur, c'est le client qui, dans un premier temps, a été la priorité.

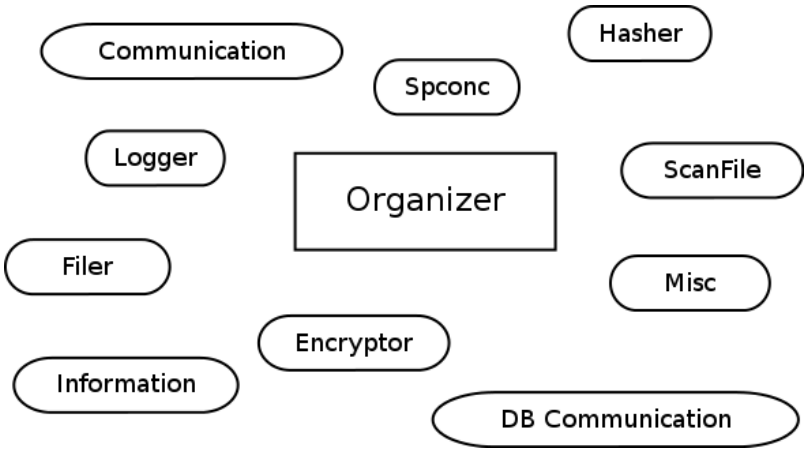


FIGURE 4.3 – Vue d'ensemble des modules.

Pour celui-ci, dix modules vont être utilisés autour d'un module central :

— **organizer :** module central qui va créer un certain nombre de *thread* et lancer les différents modules nécessaires à la sauvegarde ou à la récupération des données ;

— **information :** ce module contient les informations (méta-informations) nécessaires sur les morceaux de fichiers ainsi que sur les fichiers eux-même ;

— **filer :** ce module récupère la liste des fichiers à partir du répertoire d'entrée. Il crée les objets correspondants (module information) et les transmet au module suivant ;

— **spconc :** ce module coupe (*split*) ou concatène ce qui lui est envoyé selon qu'il s'agisse d'une sauvegarde ou d'une récupération de données ;
— **scanFile :** dans le cas où un fichier aurait déjà été sauvegardé, ce module scannera l'intégralité du fichier pour trouver les parties qui ont été modifiées et qui doivent être chiffrées et envoyées au serveur ;
— **hasher :** ce module fournit des fonctions de hachage permettant de vérifier l'intégrité des fichiers ;
— **encryptor :** ce module s'occupe du chiffrement et du déchiffrement des données ;
— **communication :** ce module s'occupe de la communication avec le serveur ;
— **dbcomm :** ce module contient les fonctions utilisées pour la communication avec la base de données ;
— **misc :** ce module contient divers outils (*miscellaneous*) pouvant être utiles dans le programme. Entre autres il contient le système de notifications réalisant la communication entre les différents modules ;
— **logger :** ce module permet de garder une trace lors de l'exécution du programme, notamment utile lors du déboguage ou si une erreur provoque l'arrêt prématuré du programme.

L'utilisation de modules doit permettre une modification aisée du programme, sans avoir à modifier tout le code. Par exemple, pour changer un algorithme, il sera seulement nécessaire d'écrire le code de celui-ci sans modifier celui déjà en place. Par la suite, il ne devrait y avoir à préciser que l'algorithme souhaité lors de la création d'un objet.

Un autre avantage de la programmation modulaire est la facilitée que cela apporte pour maintenir le code efficacement.

Dans ce but, j'utiliserai et profiterai des spécificités qu'offre la programmation orientée objet.

4.2.2 Interaction entre modules

L'utilisation de *threads* pour exécuter les différents modules complique l'interaction entre ceux-ci. Plusieurs modules sont utilisés successivement pour pouvoir traiter un fichier, de la lecture depuis le disque jusqu'à l'envoie des blocs — chiffrés — le constituant (dans le cas d'une sauvegarde). Dans ce sens, les modules doivent communiquer entre eux. Un lien est donc nécessaire pour que deux *threads* puissent s'envoyer les données liées à un fichier.

Dans un premier temps, j'avais créé un outil qui, appelé par un module, retournait un *pipe* (tunnel) soit en lecture, soit en écriture. Un autre module pouvait ensuite utiliser ce même outil pour récupérer le tunnel. Un module ayant accès au tunnel en lecture et l'autre en écriture, une communication peut être effectuée dans un sens. Un second tunnel est nécessaire pour pouvoir communiquer dans les deux sens. En triant les tunnels ainsi créés par catégories, il était possible de faire communiquer deux modules tels que, par exemple, *encryptor* et *communication*. Le premier pouvait ainsi envoyer ou recevoir des données vers ou depuis le module *communication*. Dans ce cas, la catégorie du tunnel aurait alors pu s'appeler *encomm*.

Il y avait néanmoins un problème majeur à l'utilisation d'un tel outil. S'il permettait bien d'envoyer des informations entre deux modules sans que ceux-ci ne se connaissent, il ne pouvait pas envoyer d'objet *C++*. En somme les modules ne pouvaient pas s'envoyer les méta-informations du fichier sur lequel ils étaient en train de travailler. Ils n'avaient donc que le contenu (octets) du fichiers.

Pour palier à ce problème, nous nous sommes donc dirigés vers les patrons de conception (*design pattern*).

4.2.2.1 Design pattern

Observer

Pour faire ce lien, nous avons dans un premier temps pensé au *design pattern observer* dont le principe est de notifier des objets inconnus lors d'un changement d'état. Dans notre cas, cela aurait permis que les autres modules l'observant soient avertis et puissent agir en conséquence. Cela donne également la possibilité à un module d'envoyer des informations en passant dans un certain état à des moments clés. Ces informations correspondraient aux méta-données d'un fichier, accompagnées d'un *pipe* servant de tunnel pour faire transiter le fichier en lui-même d'un *thread* à un autre.

Cette solution ne fut pas retenue car elle impliquait qu'un module ait connaissance de celui à qui envoyer les informations. Ceci n'était pas voulu afin de laisser les modules aussi indépendants que possible.

Event Notifier

Afin d'éviter que certains modules aient connaissance de l'existence d'autres modules, un système de notifications a donc été implémenté. Le principe de ce système est relativement simple à comprendre. Nous pouvons d'ailleurs trouver le même système dans la vie de tous les jours. Par exemple un journal publie régulièrement des nouvelles ayant un certain libellé (« alertes », « national », « sport », etc.). En face, il existe des utilisateurs qui peuvent, ou non, s'inscrire à un ou plusieurs type de nouvelles. Ainsi, quelqu'un inscrit aux nouvelles de type « alertes » va recevoir une notification lorsqu'une nouvelle de ce type apparaît.

Le principe utilisé dans notre cas est exactement le même : des modules lancent des notifications que d'autres reçoivent s'ils sont inscrits à celles-ci.

Pour que cela fonctionne correctement, un centre de notifications est nécessaire. Ainsi, et contrairement à un système d'« *observer* », les modules n'ont pas à avoir connaissance les uns des autres puisque chacun passe par ce centre.

Celui-ci est présenté figure 4.4. Il y a le centre de notifications *CNotificationService* qui gère les abonnements et les abonnés. Ce centre permet ainsi de notifier les modules ayant souscrit à un certain type de notification. Pour utiliser ce service, il existe deux interfaces : *ISubscriber* pour les abonnés et *INotifier* pour émettre les notifications. Ces interfaces sont donc implémentées par les différents modules utilisant le système de notifications.

La figure 4.5 montre comment le programme fonctionne dans le cas d'une première sauvegarde sur le serveur. Chaque noeud à l'intérieur du client est un module exploitant un *thread* différent. Ainsi avons-nous :

- Une seule instance de *Filer* est exécutée. Le module communique avec une des instances libres du module *Split*, i.e. qui n'est pas déjà en train de travailler sur un fichier ;
- À son tour, le module *Split* communique avec n'importe quel *thread* exécutant le module *Crypto* en lui envoyant les données à chiffrer ;
- Enfin, celui-ci envoie les données chiffrées au module *Communication* qui va simplement s'occuper de les envoyer sur le serveur.

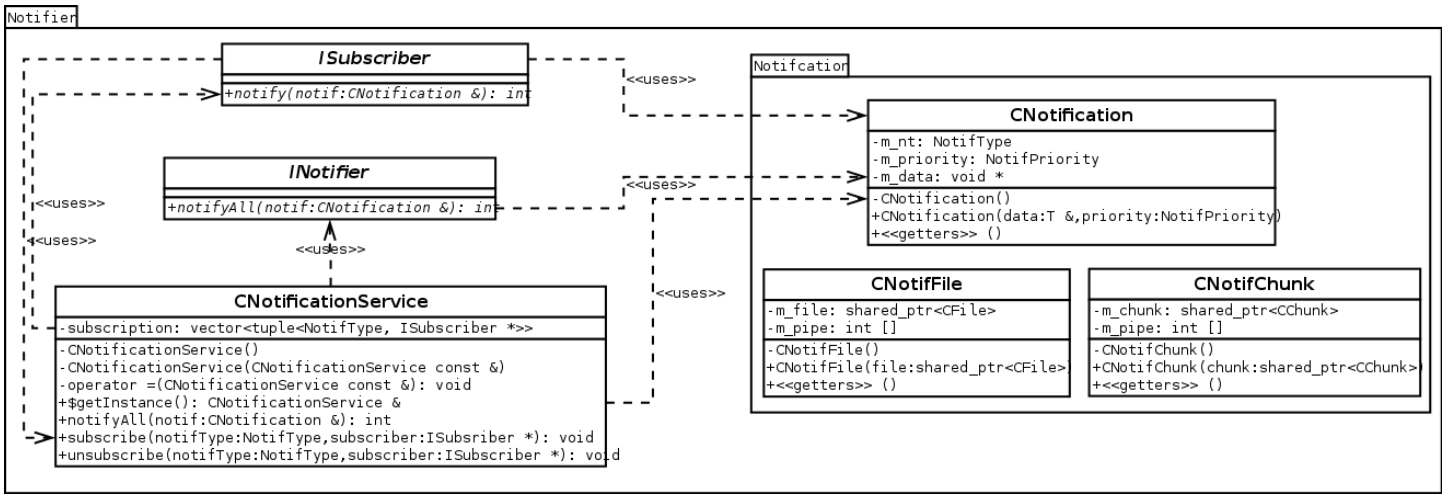


FIGURE 4.4 – Diagramme de classe — notifications.

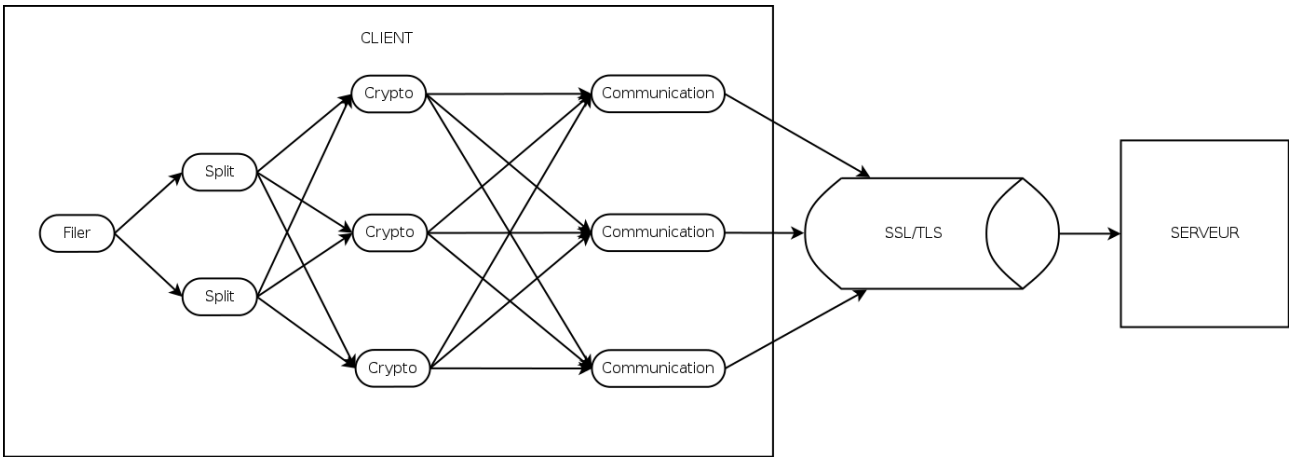


FIGURE 4.5 – Fonctionnement lors d'une première sauvegarde.

4.2.3 Base de données

L'utilisation d'une base de données semblait obligatoire à la fois du côté client et du côté serveur afin de garder les méta-données des fichiers sauvegardés mais également pouvoir garder en mémoire une trace de chaque sauvegarde effectuée. Cela donnerait la possibilité de restaurer n'importe quelle version d'un ou plusieurs fichier(s). De ce fait nous aurions donc l'ensemble des informations nécessaires à la reconstruction d'une version particulière d'un fichier.

L'utilisation d'une base de donnée côté serveur était donc évidente. Le choix d'en avoir une également sur la machine cliente s'est fait naturellement dans la mesure où cela évite d'avoir à effectuer régulièrement des connexions avec le serveur. Contrairement à la base de donnée présente sur le serveur, celle du client permettrait de ne stocker que les informations liées à la dernière sauvegarde. Ce sont en effet ces informations qui sont utiles lors d'une nouvelle sauvegarde. Le temps de traitement ainsi que la quantité de données transmises à travers le réseau Internet sont ainsi réduits.

4.2.3.1 SQL vs NoSQL

La probématique fut donc de trouver quel(s) type(s) de base de données pourrai(en)t être utilisé(s) avec d'un côté les bases de données relationnelles *SQL* et d'un autre les bases de données *NoSQL*. Les

principales différences sont exposées table 4.1.

	SQL	*NoSQL*
Stockage des données	Utilisation d'un modèle relationnel avec les traditionnelles lignes et colonnes contenant les différentes entrées de la base de données avec l'ensemble des informations correspondantes.	Il existe plusieurs modèles de stockage parmi lesquels : documents, clé-valeur, graphe, etc.
Schéma et flexibilité	Chaque table correspond à un schéma particulier. Les colonnes doivent donc être choisies à l'avance, selon les données à rentrer dans la table.	Les schémas sont dynamiques. Deux entrées d'une même table peuvent contenir différentes informations.
Adaptabilité	L'évolutivité est verticale. C'est-à-dire que plus la quantité de données augmente, plus le serveur sur lequel se trouvent les données doit être important. Ce qui peut engendrer d'important coûts	L'évolutivité est horizontale. C'est-à-dire que les données peuvent être réparties à travers plusieurs serveurs. C'est beaucoup moins onéreux que d'investir dans un très gros serveur.
Propriétés ACID [4]	La grande majorité des bases de données relationnelles ont ces propriétés	Dépend de la technologie utilisée. Plusieurs solution *NoSQL* sacrifient certaines propriétés pour de meilleures performances et une meilleure adaptabilité.

Table 4.1 – Tableau comparatif *SQL* et *NoSQL*.

4.2.3.2 Sur le serveur

Un des principaux points qui nous intéresse est ici l'adaptabilité. En effet, la quantité de données pouvant augmenter très rapidement, il peut être très coûteux d'utiliser une base de données relationnelle. Cela signifierait d'importants investissements afin de mettre en place des serveurs capables de contenir l'intégralité des données.

Le *NoSQL* était donc une solution évidente à ce problème grâce son évolutivité horizontale. De plus cette solution offre également une réplication des données à travers les différents serveurs. Celle-ci est d'ailleurs faite automatiquement sur la plupart des solutions *NoSQL*.

4.2.3.3 Sur le client

Afin d'avoir un accès rapide aux informations de la dernière sauvegarde réalisée, une base de données doit être installée sur le poste client.

Nous avons dans un premier temps pensé au modèle relationnel avec une base de données *SQLite* permettant de l'intégrer directement au programme. Celle-ci étant stockée dans un fichier, et ce indépendemment de la plate-forme utilisée, elle aurait pu correspondre à ce que voulions.

Néanmoins la quantité d'information stockée ainsi que le nombre d'accès, tant en écriture qu'en lecture, sont trop importants dès lors que plus de quelques fichiers sont sauvegardés. Il fallait donc une base de données avec de meilleurs performances que celles offertes par *SQLite*. De plus les relations entre les différentes tables étant minimes comme montrées figure 4.6, la performance du *NoSQL* pouvait être préférée par rapport aux propriétés ACID qu'offrent le *SQL*.

4. Atomicité, Cohérence, Isolation, Durabilité

Nous avons donc décidé par la suite de nous diriger, comme pour la partie serveur, sur une base de données *NoSQL*.

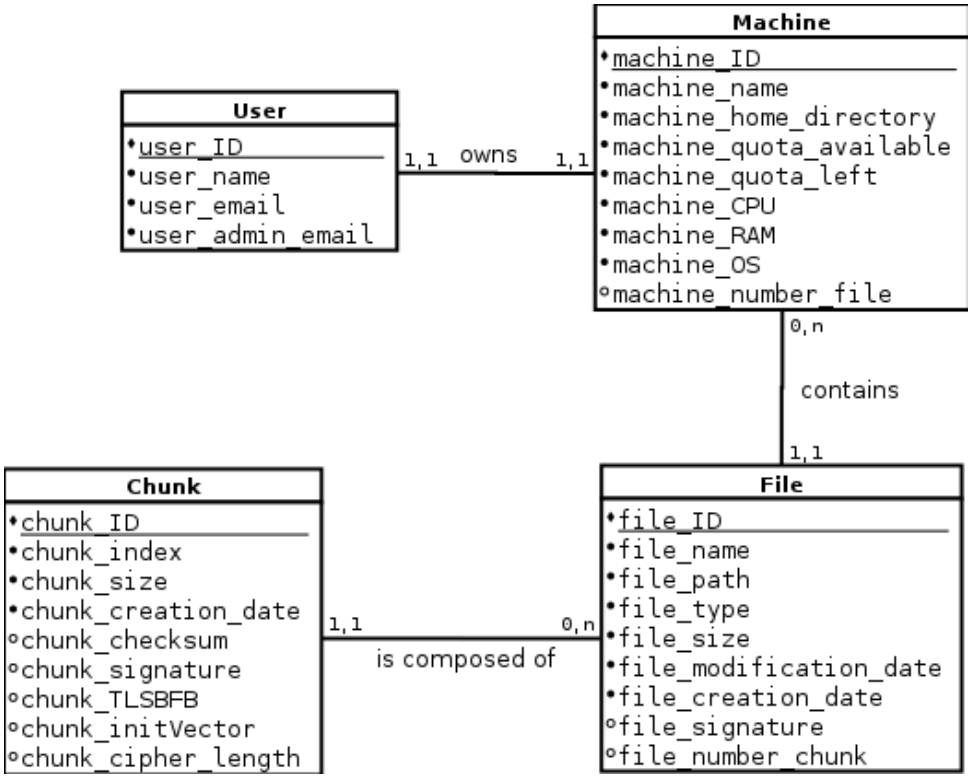


FIGURE 4.6 – Base de données relationnelle — client.

La signification des différentes colonnes présentes figure 4.6 se trouve en annexe A.

4.2.3.4 Choix de la base de données

En plus des méta-données des différents fichiers sauvegardés, le stockage des fichiers sur le serveur a influencé sur le choix de la base de données.

Certaines bases de données *NoSQL* offrent la possibilité de sauvegarder les fichiers eux-mêmes. Cela peut être intéressant que ce soit dans l'immédiat ou dans une future version de l'application.

Bien que dans un premier temps nous ayons décidé d'utiliser le système de fichier en place sur le serveur, nous avons pris en compte cette fonctionnalité.

C'est donc sur *MongoDB* que s'est arrêté notre choix. Cette base de données a plusieurs avantages qui correspondaient à ce que nous recherchions :

— Réplication automatique des données sur plusieurs serveurs ;
— Bonne performance ;
— *GridFS* pour stocker de plus grosses quantités d'informations (correspondant au fichier sauvegardé dans notre cas).

C'est également une des bases de données les plus utilisées pour du *NoSQL*. La communauté, et donc les sources pour rechercher des informations relatives à son utilisation, est assez importante. N'ayant jamais utilisé *MongoDB* auparavant il était intéressant de pouvoir assez facilement accéder à une possible aide si un problème arrivait.

Le modèle de stockage dans *MongoDB* est de type documents. La table 4.2 présente l'équivalence entre les termes utilisés pour une base de données *SQL* avec ceux utilisés dans *MongoDB*. Plus de précisions

sont disponibles sur le site *MongoDB* [13].

SQL	*MongoDB*
Table	Collection
Ligne	Document
Colonne	Champ
Index	Index
Jointure entre tables	Documents intégrés et liés

TABLE 4.2 – Équivalent des termes *SQL* avec les termes *MongoDB*.

4.2.3.5 Schémas

Bien que quelque peu différents, les schémas des bases de données présentes sur le serveur et le client ressemblent à celui présent figure 4.6.

L'absence de relations entre les différentes collections a donc obligé une modification des schémas. De plus le serveur doit tenir compte qu'un utilisateur peut avoir plusieurs machines alors que le client ne s'occupe que de la machine sur laquelle il est installé.

Client

Les structures suivantes représentent les collections de la base de données installée sur les machines clientes.

```
User :
{ "_id",
  "username",
  "email",
  "admin_email",
  "machine" :
    { "id_machine",
      "name",
      "home_directory",
      "quota_available",
      "quota_left",
      "CPU",
      "RAM",
      "OS"
    }
}
```

Il peut être intéressant de noter que l'ancienne table *Machine* a été intégrée dans la collection *User*. Ce choix s'est fait puisque, sur le client il n'y a que la machine sur laquelle le logiciel est installé qui est présente en base de données. De plus, l'accès aux données est à la fois plus simple et plus rapide

en utilisant une collection regroupant l'utilisateur et sa machine plutôt qu'en utilisant deux collections distinctes.

```
File :
{ "_id",
  "filename",
  "filepath",
  "filetype",
  "filesize",
  "modification_date",
  "creation_date",
  "signature",
  "number_chunk"
}
```

N'ayant que les informations d'un seul utilisateur et d'une seule machine, il n'est pas utile de lier d'une quelconque manière que ce soit la collection *File* avec la collection *User*.

```
Chunk :
{ "_id",
  "id_file"
  "index",
  "size",
  "creation_date",
  "checksum",
  "signature",
  "TLSBFB",
  "init_vector",
  "cipher_length"
}
```

Le lien entre la collection *Chunk* et la collection *File* est représenté par le champ *id_file* contenant l'identifiant d'un fichier auquel appartient un morceau.

Serveur

Bien que similaire, la base de données est légèrement différente sur le serveur par rapport celle du client. Les changements prennent en compte le fait qu'il puisse exister plusieurs machines pour un utilisateur mais aussi que plusieurs utilisateurs existent au sein d'une compagnie.

```
User :
{ "_id",
  "id_admin",
  "is_admin",
  "username",
  "email",
  "admin_email",
  "machine" :
```

```
    [
      { "id_machine",
        "name",
        "home_directory",
        "quota_available",
        "quota_left",
        "CPU",
        "RAM",
        "OS"
      },
      { "id_machine",
        "name",
        "home_directory",
        "quota_available",
        "quota_left",
        "CPU",
        "RAM",
        "OS"
      },
      ...
    ]
}
```

Un utilisateur pouvant avoir plusieurs machines, une liste est simplement créée, contenant l'ensemble des machines Celle-ci est intégrée dans la collection *User*. De plus, un utilisateur pouvant être un administrateur, et celui-ci pouvant gérer plusieurs utilisateurs, le lien est ajouté à l'aide du champ *id_admin*. Ainsi si le champ n'existe pas, l'utilisateur n'a pas d'administrateur au-dessus de lui. *is_admin* permet de connaître les privilèges de chaque utilisateur. L'utilisation de ces deux champs permet également de prendre compte qu'un administrateur peut lui-même être sous la supervision d'un autre administrateur. Cas qui peut être imaginable dans le cadre de l'utilisation du logiciel au sein d'une grosse société.

```
File :
{ "_id",
  "id_machine",
  "filename",
  "filepath",
  "filetype",
  "filesize",
  "modification_date",
  "creation_date",
  "signature",
  "number_chunk"
}
```

Un utilisateur pouvant possèder plusieurs machines, l'ajout du champ *id_machine* permet de pour pouvoir faire correspondre un fichier à une seule machine.

```
Chunk :
{ "_id",
```

```
  "id_file"
  "index",
  "size",
  "creation_date",
  "checksum",
  "signature",
  "TLSBFB",
  "init_vector",
  "cipher_length"
}
```

4.2.4 Stockage des fichiers

Comme nous l'avons vu précédemment, *MongoDB* combiné avec *GridFS* permet de stocker des fichiers. Néanmoins nous avons décidé d'utiliser, au moins dans un premier temps, le système de fichier disponible sur le serveur : celui-ci ayant de meilleurs performances en lecture et écriture.

La figure 4.7 présente comment les données seront stockées sur le serveur une fois sauvegardées. Ainsi chaque fichier correspond à un dossier contenant un ou plusieurs sous-dossier (un par version). La version d'un fichier correspondant à une sauvegarde à un moment donné, un dossier contient uniquement les parties modifiées du fichier.

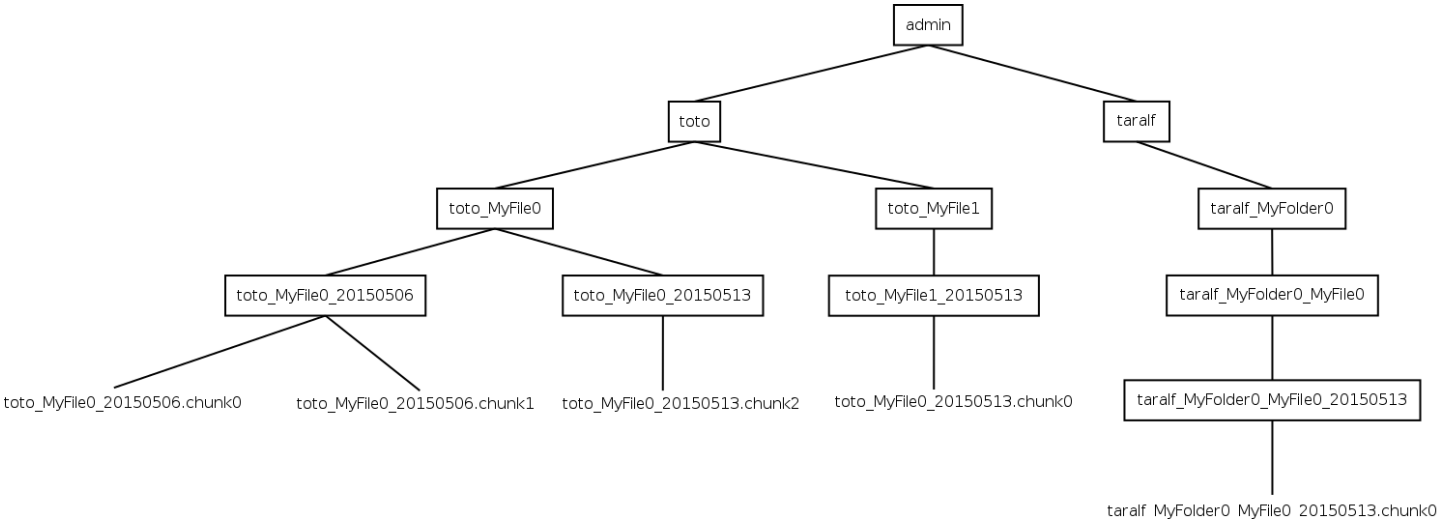


FIGURE 4.7 – Stockage des fichiers sur le serveur.

Dans notre exemple, pour récupérer le fichier *MyFile0* de l'utilisateur *toto* dans sa version datant du 13 mai 2015, nous devons donc télécharger *toto_MyFile0_20150513.chunk2* mais aussi les morceaux *.chunk0* et *.chunk1* datant de la précédente sauvegarde, au 06 mai 2015.

4.2.5 Communication client - serveur

Afin d'envoyer les fichiers à travers le réseau, un protocole doit être utilisé. Pour ce faire, il m'a été demandé d'en créer un correspondant aux données qui sont envoyées dans le cadre d'une sauvegarde. En l'occurrence, les morceaux d'un fichier avec les méta-données correspondantes.

Comme indiqué dans la section « Travail de recherche », je me suis servi du protocole créé pour *syncthings* (*BEP*). Celui-ci étant décrit sur le github du projet [16] et disponible sous licence Creative Commons [3], il est autorisé de le copier, modifier et réutiliser pour n'importe quelle utilisation, même commerciale. Bien que ce dernier point ne nous concerne pas puisque le projet sera Open Source, j'ai pu récupérer le protocole et l'adapter pour qu'il corresponde à nos besoins.

Le protocole ainsi modifié est disponible en annexe B, en anglais puisqu'également disponible sur le wiki du projet. Celui-ci n'est pas accessible à l'heure actuelle puisque le projet est toujours dans sa phase de développement.

4.2.6 Algorithme de cryptographie

À deux reprises il m'a été demandé d'utiliser des algorithmes de cryptographie. Dans un premier temps pour calculer la signature de chaque fichier et bloc constituant un fichier. Dans un second temps pour chiffrer ces blocs.

Le premier cas concerne l'utilisation d'un algorithme de hachage alors que dans le second cas, il est demandé d'utiliser un algorithme de chiffrement symétrique.

4.2.6.1 Algorithme de hachage

L'idée est d'utiliser la signature pour pouvoir d'une part vérifier l'intégrité d'un fichier, et d'autre part de repérer un bloc particulier d'un fichier d'une sauvegarde à l'autre.

C'est le second cas qui nous intéresse puisqu'en utilisant notre algorithme d'identification de bloc dans un fichier, il sera souvent demandé de calculer la signature d'un bloc. Il faut donc un algorithme qui soit à la fois assez fiable pour minimiser le risque de collision, mais rapide à calculer pour optimiser au mieux notre algorithme lors de son exécution.

SHA-256[12] semblait donc être un bon compromis. Plus lent à calculer qu'un *SHA-512*, le risque de collision est inférieur par rapport à un *SHA-128* ou *SHA-224*. Notre choix s'est donc arrêté sur cet algorithme.

4.2.6.2 Algorithme de chiffrement symétrique

David et Gaëtan avaient déjà dans l'idée d'utiliser *AES-CBC*[5] [6][1][7] avec une clé secrète de 256 bits. Cette clé secrète ne serait présente que sur la machine cliente. Ainsi, le serveur ne contiendrait que des données chiffrées, sans possibilité de les déchiffrer.

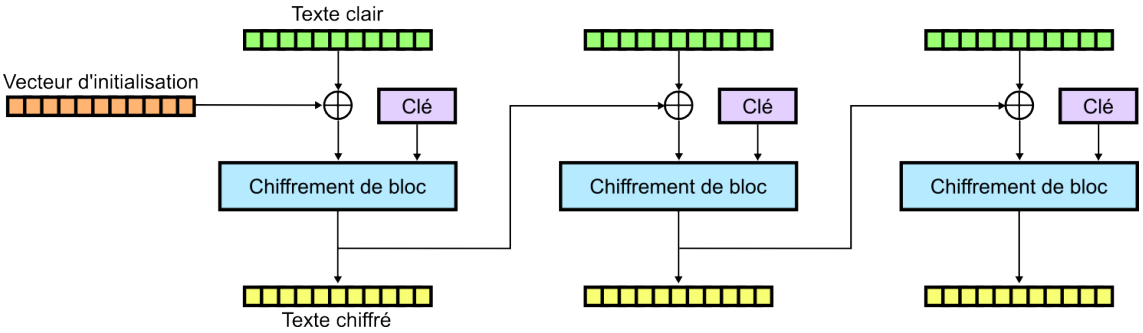


FIGURE 4.8 – Comprendre CBC.

5. AES – Advanced Encryption Standard
6. CBC – Cipher Block Chaining

La figure 4.8 présente le fonctionnement de *CBC*. L'idée est que pour chaque calcul d'un bloc de donnée le résultat chiffré soit utilisé pour calculer le bloc suivant et ainsi de suite jusqu'à ce que l'intégralité des données soit chiffrée. Un vecteur d'initialisation est donc utilisé pour chiffrer le tout premier bloc de données.

4.3 Développement

Le client et le serveur étant indépendant l'un de l'autre si ce n'est la partie communication de l'un à l'autre, le développement se fait tout aussi indépendemment.

Le client étant la plus grosse partie du système de backup, je me suis concentré sur son développement durant mon stage. Comme il avait été découpé en modules durant la phase de conception, le développement s'est fait de la même manière.

De plus chaque module a le droit à sa batterie de tests unitaires et de tests de non régression.

4.3.1 Outils de développement

Pour cette phase j'ai utilisé plusieurs outils me permettant de construire au mieux l'application demandée.

Le développement s'est principalement fait dans un environnement *Unix*. J'ai utilisé *Debian 7* pour le développement de l'application et comme outil de travail principal. Pouvant avoir accès à un environnement *Mac OS X* dans l'entreprise, j'ai également pu tester mon programme sur celui-ci.

La priorité étant d'avoir une première version fonctionnelle du programme, je n'ai pas porté celui-ci sur un système *Windows* durant mon stage.

4.3.1.1 Environnement de travail

— *vim* comme éditeur de texte ;
— *gcc* comme compilateur ;
— *gdb* pour déboguer le programme quand nécessaire ;
— *valgrind* pour tout ce qui touche à la mémoire (analyse, gestion de la mémoire) ;
— *gprof* pour ce qui s'agit de l'analyse de performance ;
— *git* comme gestionnaire de versions décentralisées.

4.3.1.2 Frameworks et bibliothèques externes

Afin de limiter les contraintes liées à l'utilisation du *framework* et des librairies externes, l'utilisation de ceux-ci était limité. Voici ceux que j'ai utilisé :

— *Boost* pour les tests ;
— *OpenSSL* pour tout ce qui est cryptographie ;
— *Magic* pour récupérer des informations sur les fichiers comme le MIME type.

En addition à cela, j'ai également utilisé le driver *C++* de *MongoDB* pour communiquer avec la base de données.

Ceux-ci sont tous disponibles pour les environnements *Mac OSX*, *Windows* et *GNU/Linux*.

4.3.2 Travail réalisé

Comme il a été initialement envisagé, j'ai développé le système de sauvegarde module par module, tous indépendants les uns des autres en préférant dans un premier temps la partie principale du programme.

Cette partie correspond à la lecture des fichiers depuis le disque, le découpage en morceaux ainsi que le chiffrement de ceux-ci.

4.3.2.1 Développement des modules

J'ai donc tout d'abord créé le *logger* qui allait m'aider, pendant l'ensemble du développement du programme, à déceler plus facilement les problèmes pouvant exister.

Ensuite, j'ai développé toute la partie concernant la cryptographie. Elle comprend les modules *hasher* et *encryptor*. J'ai utilisé les méthodes fournies par la bibliothèque *OpenSSL* afin d'écrire ces deux modules. L'utilisation de méthodes déjà existantes étant préférable à la réécriture de celles-ci. Le risque de bogues et de failles étant alors limité.

S'en est suivi le développement du module *spconc*, permettant de découper en plusieurs morceaux un fichier, ou à l'inverse de concaténer ces morceaux pour retrouver le fichier d'origine.

Ce fut ensuite au tour du module *information* d'être développé afin de pouvoir créer des objets permettant de garder les méta-données des fichiers.

À partir de là, nous avions assez de modules combinables afin de lancer le programme dans sa toute première phase d'essai. Pour simuler une sauvegarde, il manquait des prototypes pour mimer le rôle des modules *filer*, *communication* et *organizer*. En plus de cela, il nous fallait bien évidemment développer le système de notifications pour pouvoir faire communiquer les différents modules.

Une fois ceci fait, je pouvais créer l'*organizer* afin de faire fonctionner tous ces modules ensemble.

4.3.2.2 Imbrication des modules

Cette étape fut longue et beaucoup d'erreurs apparurent. Bien que chaque module fonctionnait individuellement parfaitement, l'exécution sur des threads séparés ainsi que la communication entre eux amena plusieurs problèmes.

Problèmes et solutions

Chiffrement et déchiffrement Un des points sur lequel j'ai passé le plus de temps a été la partie chiffrement et déchiffrement. Car si le programme n'émettait pas d'erreurs dans le cas du chiffrement, les morceaux créés en sortie ne correspondaient pas à ce qui pouvait être attendu.

Bien qu'ayant précisé dans le code du programme que je voulais écrire sur le disque du binaire et non des caractères ASCII, il semblait que l'utilisation des *iofstream* (flux d'entrée/sortie sur des fichiers) posait problème. En effet, un nombre aléatoire de caractères était écrit dans les fichiers de sortie.

Il était donc tout simplement impossible de reconstruire les fichiers proprement puisqu'il était impossible alors de les déchiffrer.

La solution a été l'utilisation des *FILE*[7] du *C* pour tout ce qui est gestion de fichiers en lieu et place des *iofstream*[8] du *C++*.

Problèmes de mémoire Avec l'utilisation du *C++*, je devais gérer moi-même l'utilisation de la mémoire. J'ai donc rencontré deux soucis : fuites de mémoire et libérations précipitées de la mémoire.

Dans le premier cas, il suffisait simplement de libérer la mémoire au bon moment. C'est-à-dire lorsqu'il n'est plus utile d'accéder à la mémoire allouée précédemment dans le programme.

7. Structure permettant de contenir les informations nécessaires au contrôle d'un flux [4]
8. Classes (*ifstream* [6] et *ofstream* [9]) gérant des flux en entrée ou en sortie pour travailler sur des fichiers

En ce qui concerne le second cas, les libérations précipitées de la mémoire posaient problème puisque plusieurs *threads* tournaient en parallèles. C'était alors au *thread* ayant alloué la mémoire de la libérer. Néanmoins, d'autres modules, exécutés sur d'autres *threads*, pouvaient utiliser cette même mémoire. C'était notamment le cas des objets contenant les méta-informations des fichiers. Si le premier module libérait la mémoire allouée, il devenait impossible pour les suivants d'utiliser cette mémoire qui ne contenait alors plus l'objet voulu.

Pour résoudre ce problème deux solutions s'offraient à moi :

— La première était de libérer la mémoire une fois que tous les modules l'utilisant avaient fini leur exécution sur l'objet contenu dans cet espace mémoire ;

— La seconde était d'utiliser des pointeurs intelligents. Ceux-ci fonctionnent de la même façon qu'un *garbage collector* (ramasse-miettes) [9] : tant que la mémoire est utilisée, elle n'est pas libérée ; elle est libérée automatiquement lorsqu'elle n'est plus utilisée dans aucun module.

La seconde solution était bien évidemment préférable puisqu'elle réglait aussi les soucis de fuites de mémoire pouvant exister çà et là dans le programme.

Ajout des fonctionnalités

Une fois toutes les erreurs corrigées, nous avions une version fonctionnelle du programme. Cette version permettait de chiffrer un fichier (que l'on précisait dans un premier temps en début d'exécution) ainsi que de le déchiffrer. Les morceaux du fichier ainsi créés et chiffrés étaient stockés en local sur le disque dur afin de « simuler » le module *communication*.

À partir de là il ne restait plus qu'à rajouter les différentes fonctionnalités nécessaires au bon fonctionnement du programme tel qu'il avait été demandé.

J'ai donc tout d'abord ajouté la possibilité de lire un dossier de manière récursive au lieu de lire uniquement un simple fichier. Quelques petites erreurs que je n'avais pas détecté lors de la précédente version du programme sont apparues. Entre autres, et principalement, tous les descripteurs de fichiers [10] ouvert n'étaient pas forcément fermés. Cela amenait le programme à planter à cause d'un trop grand nombre de descripteurs ouverts. J'ai donc du les fermer pour que le programme puisse s'exécuter sans problèmes.

À ce moment j'ai testé le programme sur *Mac OSX*. Une erreur de liaison avec les bibliothèques utilisées empêchait la compilation. Une simple modification du *Makefile* [11] a permis de résoudre ce léger souci. Le programme pouvait ainsi tourner sur le système d'exploitation d'*Apple*.

Travail à venir

L'étape suivante consistait à ajouter la base de données, nécessaire pour pouvoir intégrer au programme l'algorithme trouvé en tout début de stage (voir section « Travail de recherche »).

Ne connaissant pas *MongoDB*, je me suis donc formé pour comprendre comment celui-ci fonctionnait ainsi que pour l'utiliser dans un programme en *C++*. De plus, un cours en ligne commençait gratuitement sur le site « *University* » de *MongoDB*[8]. J'ai suivi ce cours qui montrait et expliquait comment utiliser *MongoDB* pour un développeur. Même si le langage utilisé dans le cours était *Python*, le principe restait

9. Sous-système informatique de gestion automatique de la mémoire. Il est responsable du recyclage de la mémoire préalablement allouée puis inutilisée.
10. Un descripteur de fichier est une clé abstraite pour accéder à un fichier.
11. Fichier(s) permettant de compiler et lier un programme avec les bibliothèques utilisées.

le même et j'ai pu réutiliser ce que j'apprenais dans ce cours en adaptant pour le projet de système de sauvegarde.

Au moment de rédiger le rapport j'en étais donc à cette étape là. Une fois la base de données ajoutée ainsi que le prototype de l'algorithme intégré dans le programme, il ne restera plus qu'à détecter quels fichiers ont été modifiés depuis la précédente sauvegarde avant de pouvoir faire tourner le programme pour ne prendre en compte que ces fichiers là et les analyser, ainsi que les fichiers nouvellement créés.

4.3.3 Tests

Une suite de test unitaires et de non régression ont été écrites pour chaque module. Ainsi est faite la vérification de chaque méthode de chaque classe de chaque module, dans la mesure du possible. En effet, et c'est notamment vrai pour ce qui touche aux *threads*, il est parfois difficile d'écrire des tests permettant de couvrir l'ensemble des cas pouvant exister.

Afin de réaliser ces tests, j'ai utilisé le framework de test *Boost*. Celui-ci m'a de ce fait permis de valider le bon fonctionnement des différents éléments constituant le programme.

Ces tests ont été écrits au fur et à mesure de l'avancement du projet. De plus, pour chaque test, une vérification de l'utilisation de la mémoire était effectuée à l'aide de l'outil *valgrind*. Cela me permettait de vérifier l'éventuelle présence de fuites de mémoire. Je pouvais par la suite apporter les modifications nécessaires au programme.

Un module ainsi écrit devait passer l'ensemble des tests écrits pour celui-ci avant que je puisse passer à l'écriture du module suivant. Ainsi, lorsque j'écrivais un module, je savais que l'ensemble des modules déjà créés fonctionnaient tels qu'ils avaient été conçus et ce, sans fuites de mémoire.

5 Conclusion

Durant mon stage j'ai pu mettre en pratique l'enseignement reçu durant mes cinq années d'étude, que ce soit à l'IUT d'Aix-en-Provence, à la faculté des sciences de l'université d'Aix-Marseille ou à l'université d'Uppsala.

Le plan de restructuration effectué au sein de l'entreprise a été source d'expérience même si, étant stagiaire, cela ne m'a pas autant affecté que les autres employés de la compagnie. Néanmoins, il est représentatif de ce qu'il se passe dans de nombreuses startup mais aussi de ce qu'il peut se passer dans des entreprises de taille plus importante.

De plus ce stage a été pour moi l'occasion de travailler avec des personnes de culture et de pays différents. Faisant ce stage à l'étranger, j'ai donc régulièrement dû m'adapter à la culture locale, que ce soit durant mes heures de travail ou en dehors de celles-ci.

Bien qu'un contrat en CDI était originalement prévu, sous réserve que l'entreprise veuille me garder et que j'accepte de rester, celui-ci n'était plus d'actualité à la fin du stage suite à la restructuration effectuée.

Ce stage de fin d'étude m'a donc permis d'acquérir de l'expérience tant au niveau du développement d'un produit que sur celui de la vie d'une startup.

Table des figures

2.1 Lieux où trouver Arksens. 8
2.2 Beau Plan Business Park 11

3.1 Page d'accueil du *manager* 13

4.1 Schéma initial du fonctionnement du backup 15
4.2 Schéma montrant brièvement l'algorithme *rsync*. 17
4.3 Vue d'ensemble des modules. 19
4.4 Diagramme de classe — notifications. 22
4.5 Fonctionnement lors d'une première sauvegarde. 22
4.6 Base de données relationnelle — client. 24
4.7 Stockage des fichiers sur le serveur. 28
4.8 Comprendre CBC. 29

Liste des tableaux

2.1 Produits et solutions par *Arksens* 9

4.1 Tableau comparatif *SQL* et *NoSQL*. 23
4.2 Équivalent des termes *SQL* avec les termes *MongoDB*. 25

A.1 Information colonnes – User. 39
A.2 Information colonnes – Machine. 39
A.3 Information colonnes – File. 40
A.4 Information colonnes – Chunk. 40

Bibliographie

[1] Advanced Encryption Standard. https://fr.wikipedia.org/wiki/Advanced_Encryption_Standard.

[2] AlpineDC. http://www.alpinedc.ch/en/.

[3] Creative Commons Attribution 4.0 International License. http://creativecommons.org/licenses/by/4.0/.

[4] FILE. http://www.cplusplus.com/reference/cstdio/FILE/.

[5] Highlights of the USA PATRIOT Act. http://www.justice.gov/archive/ll/highlights.htm.

[6] ifstream. http://www.cplusplus.com/reference/fstream/ifstream/.

[7] Mode d'opération (cryptographie). https://fr.wikipedia.org/wiki/Mode_d%27op%C3%A9ration_%28cryptographie%29#Encha.C3.AEnement_des_blocs_:_.C2.AB_Cipher_Block_Chaining_.C2.BB_.28CBC.29.

[8] MongoDB University. https://university.mongodb.com/.

[9] ofstream. http://www.cplusplus.com/reference/fstream/ofstream/.

[10] Open Source Initiative. http://opensource.org/.

[11] OVH. https://www.ovh.com/fr/.

[12] SHA-2. https://fr.wikipedia.org/wiki/SHA-2.

[13] SQL to MongoDB Mapping Chart. http://docs.mongodb.org/manual/reference/sql-comparison/.

[14] Syncthing. https://syncthing.net/.

[15] Arksens, 2015. https://arksens.com.

[16] J. Borg et S. Dapper : Block Exchange Protocol v1. https://github.com/syncthing/specs/blob/master/BEPv1.md.

[17] A. Tridgell et P. Mackerras : rsync. https://rsync.samba.org/.

[18] A. Tridgell et P. Mackerras : The rsync algorithm, Nov. 1998. https://rsync.samba.org/tech_report/tech_report.html.

Deuxième partie

Annexes

A | Base de données relationnelle

Annexe A.1 : User

Colonne	Information
user_id	Identifiant unique pour l'utilisateur
user_name	Nom de l'utilisateur
user_email	Email de l'utilisateur
user_admin_email	Email de l'administrateur en charge de l'utilisateur. Utilisé pour prévenir automatiquement l'administrateur en cas de problème de fonctionnement.

Table A.1 – Information colonnes – User.

Annexe A.2 : Machine

Colonne	Information
machine_id	Identifiant unique pour la machine
machine_name	Nom de la machine
machine_home_directory	Répertoire racine des fichiers à sauvegarder
machine_quota_available	Quota total disponible pour la machine
machine_quota_left	Quota restant pour la machine
machine_CPU	Informations relatives au processeur utilisé
machine_RAM	Quantité de RAM disponible
machine_OS	Système d'exploitation utilisé
machine_number_file	Nombre de fichiers sauvegardés

Table A.2 – Information colonnes – Machine.

Annexe A.3 : File

Colonne	Information
file_id	Identifiant unique du fichier
file_name	Nom du fichier
file_path	Chemin du fichier en local
file_type	Type du fichier
file_size	Taille du fichier en octet
file_modification_date	Date de dernière modification
file_creation_date	Date de création
file_signature	Signature numérique
file_number_chunk	Nombre de morceaux composant le fichier

TABLE A.3 – Information colonnes – File.

Annexe A.4 : Chunk

Colonne	Information
chunk_id	Identifiant unique d'un morceau de fichier
chunk_index	Index du morceau de fichier
chunk_size	Taille du morceau en octet
chunk_creation_date	Date de création
chunk_checksum	Checksum
chunk_signature	Signature numérique
chunk_TLSBFB [1]	Deux bits de poids faible du premier octet du morceau. Utilisés pour répartir le travail entre plusieurs threads lors du scan d'un fichier pour trouver les parties modifiées de celui-ci.
chunk_init_vector	Vecteur d'initialisation utilisé pour l'algorithme de chiffrement AES
chunk_cipher_length	Taille du morceau chiffré en octet

TABLE A.4 – Information colonnes – Chunk.

1. Two Last Significant Bit of First Byte

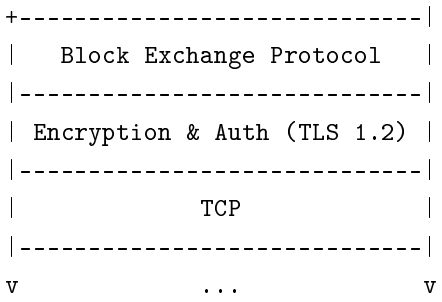
B | Protocole de communication

Annexe B.1 : Definition

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

Annexe B.2 : Transport and Authentication

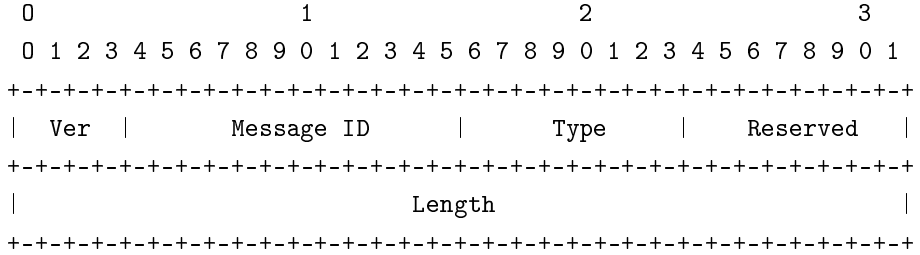
This protocol is deployed as the highest level in a protocol stack, with the lower level protocols providing additional encryption and authentication.



Annexe B.3 : Messages

Every message starts with one 32 bit word indicating the message version, type and ID, followed by the length of the message. The header is in network byte order, i.e. big endian.

— Message Structure :



For version 1 the Version field is set to zero. Future versions with incompatible message formats will increment the Version field. A message with an unknown version is a protocol error and MUST result

in the connection being terminated. A client supporting multiple versions MAY retry with a different protocol version upon disconnection.

The Message ID is set to a unique value for each transmitted request message. In response messages it is set to the Message ID of the corresponding request message. The uniqueness requirement implies that no more than 4096 messages may be outstanding at any given moment. The ordering requirement implies that a response to a given message ID also means that all preceding messages have been received, specifically those which do not otherwise demand a response. Hence their message ID :s may be reused.

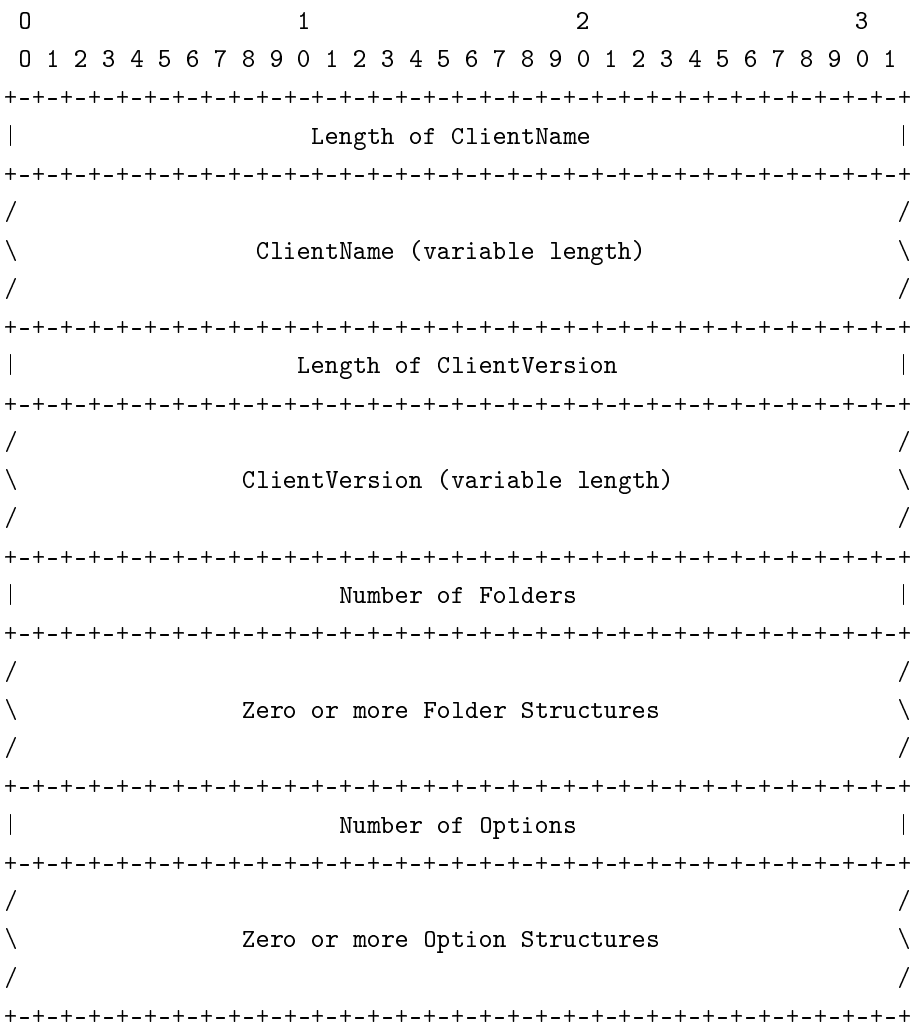
The Type field indicates the type of data following the message header and is one of the integers defined below. A message of an unknown type is a protocol error and MUST result in the connection being terminated.

Annexe B.4 : Types

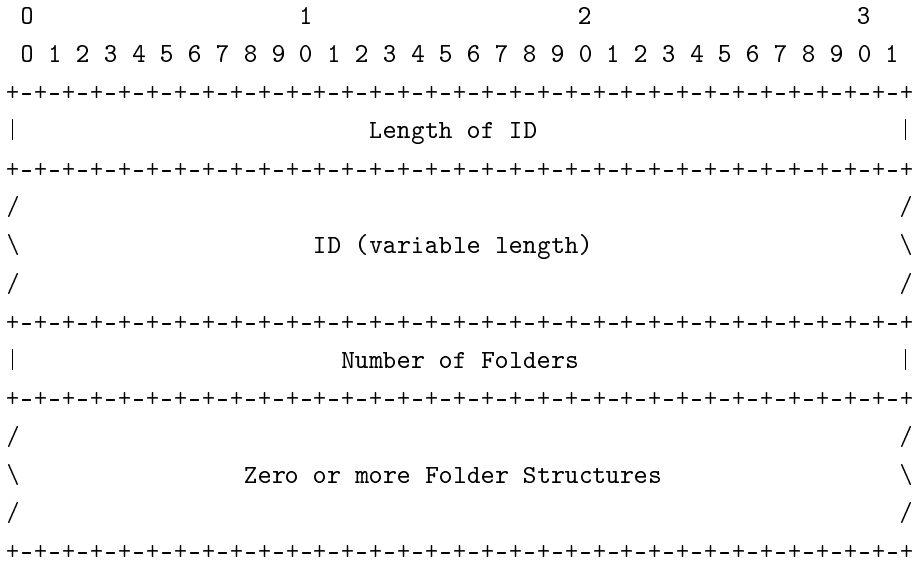
Annexe B.4.1 : 0 - Client Config

This informational message provides information about the client configuration as it pertains to the current connection. A Client Config message MUST be the first message sent on a connection. Additional Client Config messages MUST NOT be sent after the initial exchange.

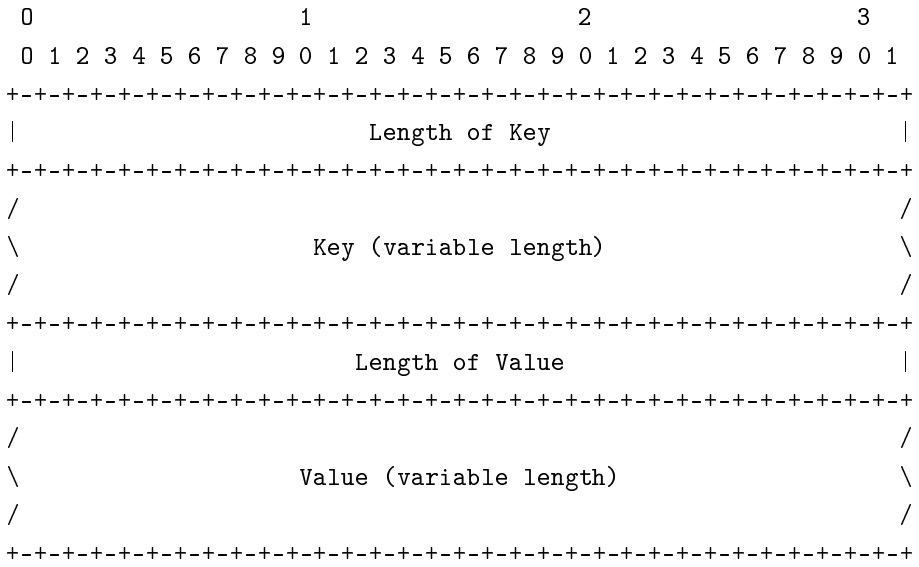
— ClientConfigMessage Structure :



— Folder Structure :



— Option Structure :



The ClientName and ClientVersion fields identify the implementation. The values SHOULD be simple strings identifying the implementation name, as a user would expect to see it, and the version string in the same manner. An example ClientName is "acs" and an example ClientVersion is "v0.7.2". The ClientVersion field SHOULD follow the patterns laid out in the [Semantic Versioning](http ://semver.org/) standard.

The Folders field lists all folders that will be synchronized over the current connection. Each folder has a list of sub-folders.

The Options field contain option values to be used in an implementation specific manner. The options list is conceptually a map of Key => Value items, although it is transmitted in the form of a list of (Key, Value) pairs, both of string type. Key ID :s are implementation specific. An implementation MUST ignore unknown keys. An implementation MAY impose limits on the length keys and values. The options list may be used to inform of relevant information such as quota and other user information for example.

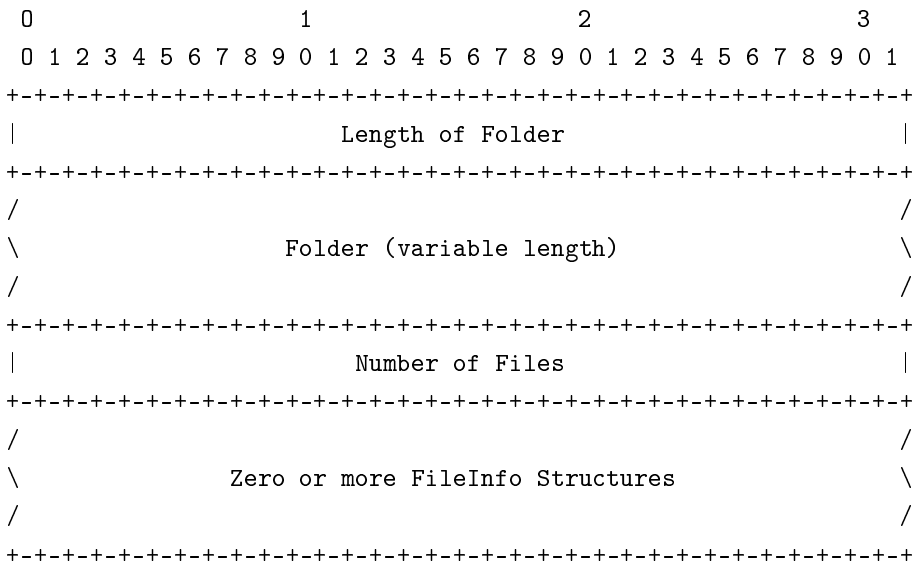
Annexe B.4.2 : 1 - Index && 7 - Index Update

This TypeMessages will be used for Client -> Server communication, i.e. the actual backup.

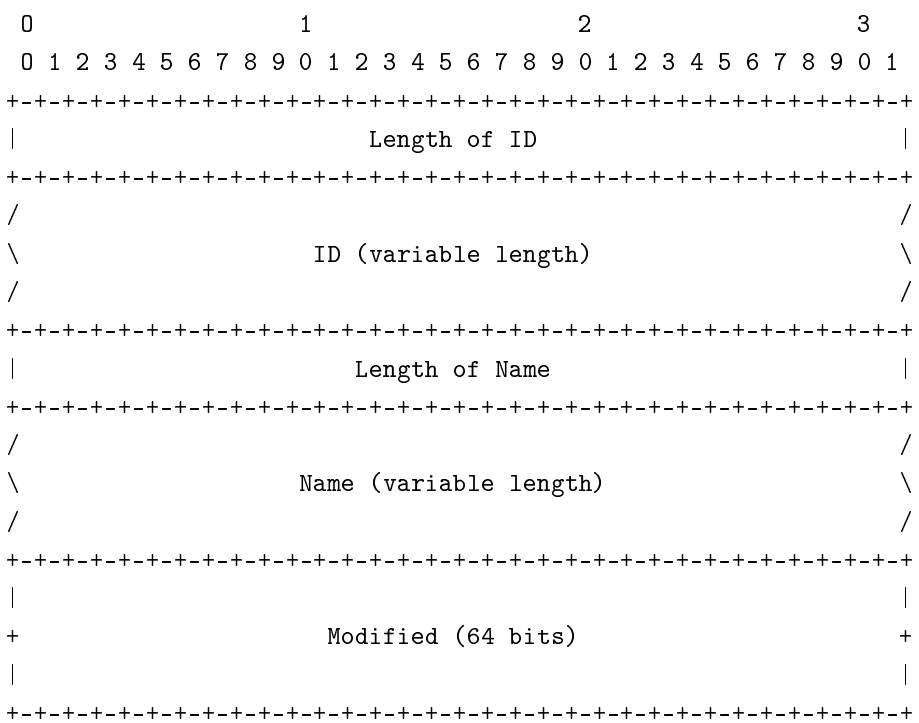
The Index and Index Update messages define the contents of the senders folder. An Index message represents the full contents of the folder and thus supersedes any previous index. An Index Update amends an existing index with new information, not affecting any entries not included in the message.

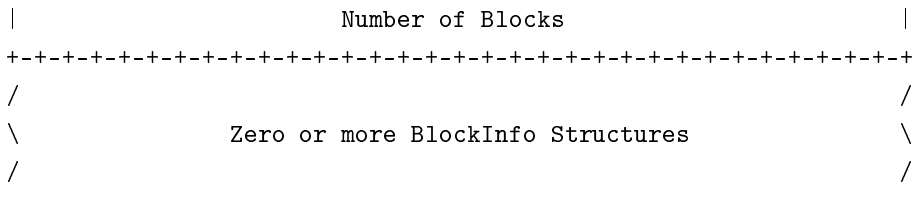
An Index or Index Update message MUST be sent for each folder included in the Client Config message, and MUST be sent before any other message referring to that folder. If the folder contents change from non-empty to empty, an empty Index message MUST be sent. There is no response to the Index message.

— IndexMessage Structure :

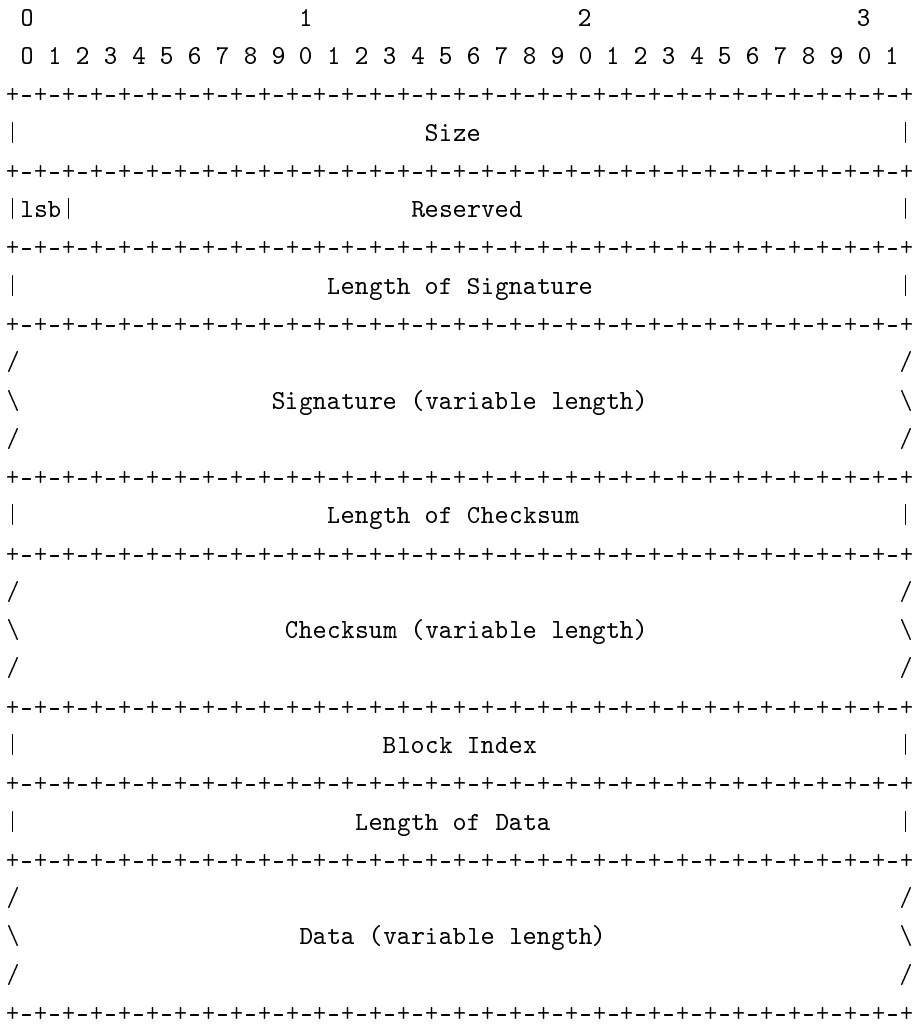


— FileInfo Structure :





— BlockInfo Structure :



The Folder field identifies the folder that the index message pertains to.

The Name is the file name path relative to the folder root. Like all strings, the Name is always in UTF-8 NFC regardless of operating system or file system specific conventions. The Name field uses the slash character ("/") as path separator, regardless of the implementation's operating system conventions. The combination of Folder and Name identifies each file on a Client machine as well as a unique ID specified by Adhara.

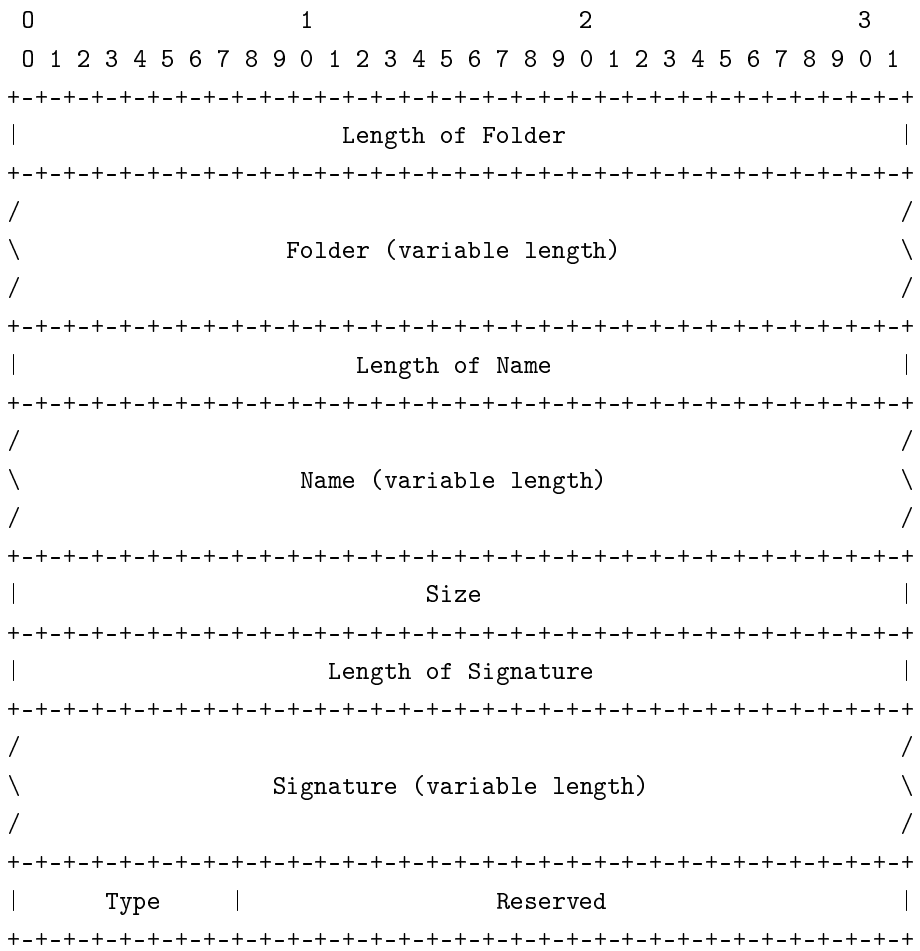
The Modified time is expressed as the number of seconds since the Unix Epoch (1970-01-01 00 :00 :00 UTC).

The Blocks list contains the size, signature, checksum, block index and the two least significant bit of the first byte of the block (lsb) for each block in the file.

Annexe B.4.3 : 2 - Request

The Request message expresses the desire to retrieve a file from the server or to get the data on modified files to synchronize.

— RequestMessage Structure :



The size and signature field MAY be set to the expected size and signature values of the block, or may be left empty (zero length). If set, the server SHOULD ensure that the transmitted block matches the requested size and signature.

Type corresponds to the expected type of the reply :

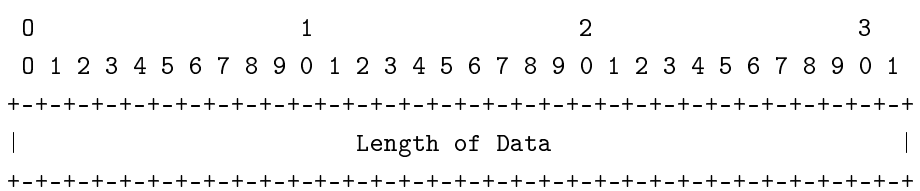
— 3 - ResponseData

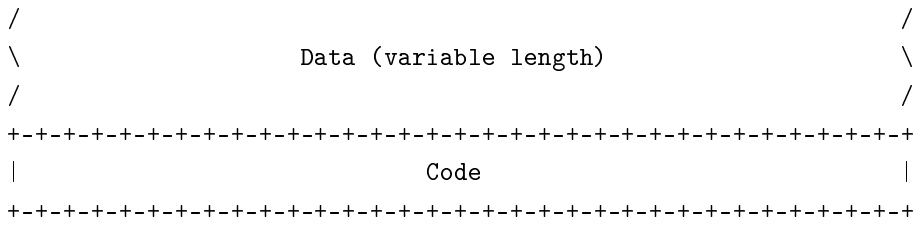
— 4 - ResponseFileInfo

Annexe B.4.4 : 3 - ResponseData

The ResponseData message is sent in response to a Request message.

— ResponseDataMessage Structure :





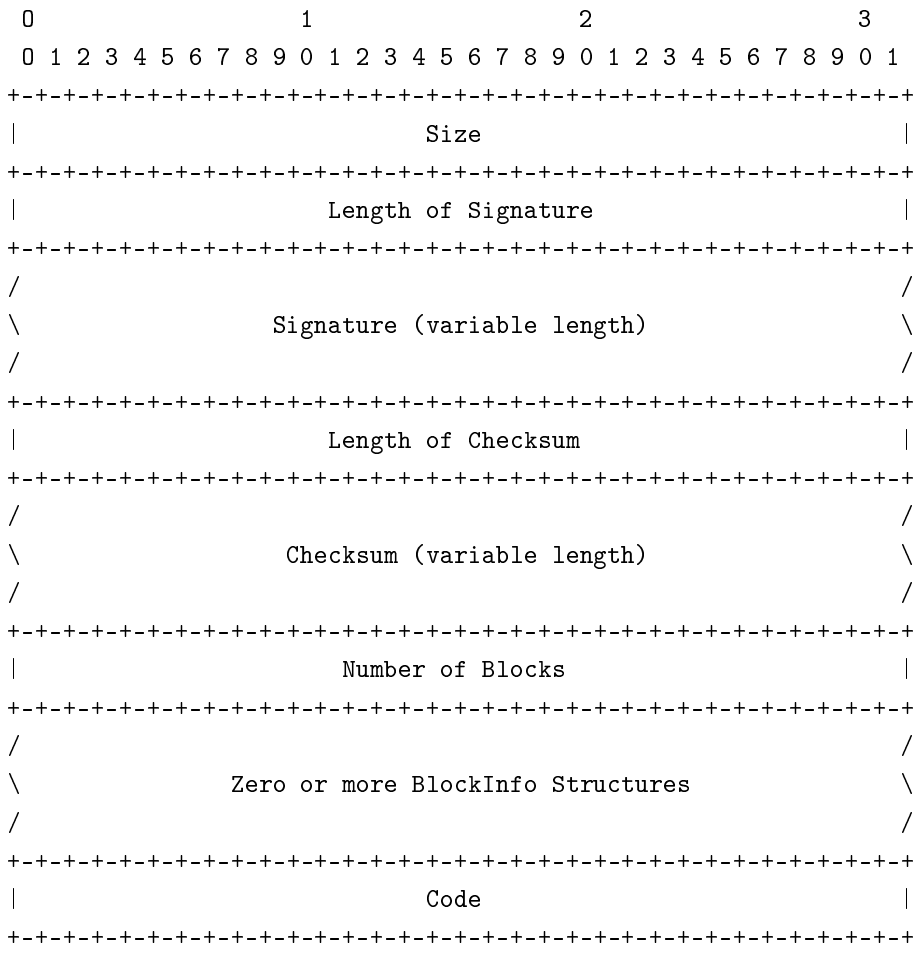
The Code field contains an error code describing the reason a Request could not be fulfilled, in the case where a zero length Data was returned. The following values are defined :

— 0 : No Error (Data should be present)
— 1 : Generic Error
— 2 : No Such File (the requested file does not exist, or the offset is outside the acceptable range for the file)
— 3 : Invalid (file exists but has invalid bit set or is otherwise unavailable)

Annexe B.4.5 : 4 - ReponseFileInfo

The ResponseFileInfo is sent in response to a Request message.

— ResponseFileInfoMessage Structure :



— BlockInfo Structure :

The different fields are defined as they were previously.

Annexe B.4.6 : 5 - Ping

No content. The Ping message is used to determine that a connection is alive.

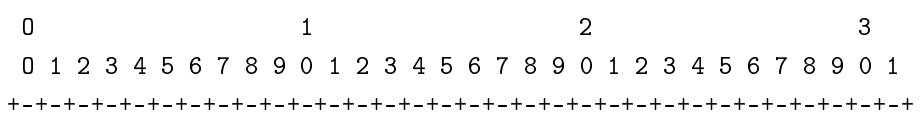
Annexe B.4.7 : 6 - Pong

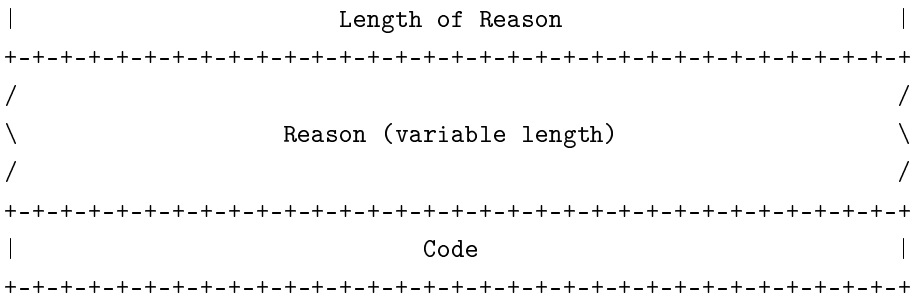
No content but copies the Message ID from the Ping.

Annexe B.4.8 : 8 - Close

The Close message MAY be sent to indicate that the connection will be torn down due to an error condition. A Close message MUST NOT be followed by further messages.

— CloseMessage Structure :





The Reason field contains a human description of the error condition, suitable for consumption by a human. The Code field is for a machine readable error code.