Telecommunication - option réseaux, sécurité et objet connecté

Alexandre Olivié
alexandre.olivie@bordeaux-inp.fr

Interopérabilité, cryptographie et télécommunication de la première station de surveillance de la pollution lumineuse au panama

Rapportrice : Hasnaa Aniss
hasnaa.aniss@univ-eiffel.fr

Tuteur de stage : José Robles
jrobles@indicatic.org.pa

INDICATIC AIP
Instituto Nacional de Investigaciones Científicas Avanzadas en Tecnologías de Información y Comunicación AIP

2025

Table des matières

1 Introduction **5**

2 Présentation de l'entreprise **7**

3 Contexte **9**

4 Analyse de l'existant **10**

5 Cahier des charges **11**
5.1 Vue d'ensemble 12

6 INDI **14**
6.1 Fonctionnement 14

7 Partie 1 : De l'éxécution du code jusqu'au moment d'avant l'envoie **16**
7.1 Diagramme général et explication 16
7.2 Exécution de main.c et traitement des données de chaque équipement (a) 17
7.3 Fonction execution, Séquenceur (b) 18
7.4 Fonction Maketram, organisation des métadonnées (c) 19
7.5 Fonction Log_NAS, information de connexion (d) 19
7.6 Fonction Connect_To_NAS, transfert des données vers le NAS (e) 20
7.7 Fonction ChecksumGenerate, réalisation des empreintes numériques (f) 22

8 Partie 2 : Chiffrement des empreintes numériques **23**
8.1 Diagramme général et explication 23
8.2 Première étape : partage de clé secrète de Shamir 23
8.3 Deuxième étape : César généralisé appliqué à une matrice numérique 24
8.4 Troisième étape : Application d'un masque binaire 25

9 Partie 3 : Transfert des données **28**
9.1 Raisons ayant conduit à l'adoption du protocole SFTP 28
9.1.1 Benchmarking Download et Upload SFTP/FTPS 29
9.2 Tailscale 30

10 Partie 4 : Traitement des données lors de leur réception **32**
10.1 Diagramme général et explication 32
10.2 Partie a : Recherche de fichier CryptFile existant 32
10.3 Partie b : Récupération des valeurs du fichier CryptFile.txt 33
10.4 Partie c : Réalisation du déchiffrement 34
10.5 Partie d : Organisations des fichiers 34

11 Partie 5 : Déchiffrement des empreintes numériques **35**
11.1 Diagramme général et explication 35
11.2 Première étape : Opération inverse du XOR 35
11.3 Deuxième étape : Opération inverse de César 35
11.4 Troisième étape : Interpolation lagrangienne 36

12 Partie 6 : Interface, lancement et contrôle de la station **38**
12.1 Interface et lancement 38
12.2 Contrôle de la station 39

13 Transfert de savoir **40**

14 Diagramme de Gantt et KPI (Key Performance Indicator) **41**
14.1 Explication du diagramme de gantt et point de vue sur l'organisation du temps 41
14.2 KPI 41
14.3 Conclusion sur la gestion de projet 42

15 Conclusion **44**
15.1 Les perspectives futures du projet 44

16 Références **45**

Liste des figures

1 Logo de INDICATIC . . . 7
2 Édifice des Laboratoires de Recherche et d'Innovation . . . 7
3 Organigramme du Conseil d'Administration et des effectifs de INDICATIC AIP . . . 8
4 Flux des traitements entre station et NAS . . . 11
5 Vue d'ensemble du fonctionnement du code d'interopérabilité . . . 13
6 Résultat commande indiserver . . . 15
7 Résultat commande indi_getprop . . . 15
8 Diagramme Partie 1 . . . 16
9 Diagramme partie a . . . 17
10 Déroulement du script alpy.sh . . . 17
11 Extrait du script alpy.sh concernant la figure 10 . . . 18
12 Diagramme Partie b . . . 18
13 Exemple nommage fichier info . . . 19
14 Script de la fonction Maketram . . . 19
15 Script de la fonction Log_NAS . . . 19
16 Diagramme Partie e . . . 20
17 Organisation fichier *count.txt*, cas aucun fichier existant . . . 21
18 Organisation fichier *count.txt*, cas fichier existant . . . 21
19 ChecksumGenerate fonction . . . 22
20 Diagramme Partie 2 . . . 23
21 Script de la fonction caesar . . . 24
22 Traitement des données jusqu'à la deuxième étape . . . 25
23 Script de la fonction xor . . . 26
24 Application du masque binaire . . . 26
25 Traitement XOR jusqu'à écriture dans le fichier CryptFile.txt . . . 27
26 Fichier CryptFile.txt . . . 27
27 Vue d'ensemble du transfert des données . . . 28
28 Script de l'envoie des fichiers . . . 28
29 Comparaison vitesse de transferts des CLI lftp et curl, capture d'écran réalisée à partir du site [sftptogo.com], consulté le [13 Mars 2025]. . . . 29
30 Envoie de différentes tailles, capture d'écran réalisée à partir du site [sftptogo.com], consulté le [13 Mars 2025]. . . . 30
31 Interface de Tailscale, capture d'écran réalisée à partir du site [tailscale.com], consulté le [17 Mars 2025]. . . . 31
32 Diagramme Partie 4 . . . 32
33 Diagramme Partie a . . . 33
34 Vérifications des empreintes numériques . . . 34
35 Diagramme Partie 5 . . . 35
36 Traitement des données déchifrement . . . 37
37 Diagramme Partie 6 . . . 38
38 Organisation des tâches dans Crontab . . . 39
39 Organisation GitHub . . . 40
40 Capture d'écran du dépôt GitHub . . . 40
41 KPI . . . 41
42 Diagramme de Gantt . . . 43

Liste des tableaux

1 Rôles des composants dans l'architecture INDI 14
2 Correspondance entre les types de demande et les propriétés INDI 15
3 Tableau XOR 26
4 Comparaison entre les protocoles FTP, FTPS et SFTP 30

Abstrait

Ce rapport présente le développement et le fonctionnement d'un code d'interopérabilité conçu dans le cadre d'un projet inédit au Panama dédié à l'étude de la pollution lumineuse, l'interopérabilité signifie la capacité de différents systèmes, appareils ou logiciels à échanger, comprendre et utiliser mutuellement leurs données et leurs services, même s'ils ont été conçus de manière indépendante ou avec des technologies différentes. L'accent est mis sur le transfert fluide, sécurisé et intègre des données capturées par les différents appareils de mesure, vers un NAS (Network Attached Storage) hébergé dans les locaux de l'INDICATIC (Instituto de Innovación, Investigación y Desarrollo de las Tecnologías de la Información y las Comunicaciones.) AIP. Chaque section du rapport détaille les mécanismes mis en œuvre, afin de permettre une compréhension globale du fonctionnement du code et des équipements impliqués dans le projet avec à la fin la démonstration des résultats obtenus ainsi que la gestion du projet. À la date de rédaction, dans la région centraméricaine, il n'existe pas encore de réseau de suivi de la pollution lumineuse, bien que des efforts notables soient observés dans plusieurs pays développés. Il convient également de mentionner un travail significatif sur la pollution lumineuse réalisé à La Réunion (latitudes tropicales), qui constitue une référence importante dans ce domaine.

1 Introduction

Ce rapport présente le travail effectué durant un stage de 24 semaines réalisé en troisième année d'études d'ingénieur au sein du laboratoire INDICATIC situé au Panama. Dans ce contexte, l'informatique joue un rôle clé en facilitant l'intégration des nouvelles technologies au sein des systèmes existants, notamment pour le traitement des données complexes et leur interopérabilité entre différentes stations. L'objectif principal de ce projet est de développer un code permettant le transfert de données hétérogènes entre différents appareils de manière interopérable. Le système devra gérer efficacement une grande quantité de données — environ 120 Go par nuit et par station en assurant un transfert fluide vers le système de stockage situé dans les locaux de l'INDICATIC AIP. Cette étude est essentielle pour garantir l'efficacité et la fiabilité des systèmes de transmission de données dans des environnements à haute capacité. Pour atteindre cet objectif, plusieurs aspects du système devront être abordés, incluant la **réception des données des équipements**, leur **chiffrement**, leur **transfert**, leur **réception et traitement dans le système de stockage**, ainsi que la mise en place de méthodes pour garantir l'autonomie du système. Le développement de ce code se fonde sur l'utilisation de solutions techniques spécifiques, telles que le logiciel **Tailscale** pour la mise en réseau, et une méthode de chiffrement et de déchiffrement rigoureuse pour assurer la sécurité et l'intégrité des données. Le rapport se structure en six sections principales. Chaque section est dédiée à un aspect spécifique du développement du code d'interopérabilité, allant de l'analyse des besoins à la mise en œuvre des solutions techniques. À travers ce processus, il est analysé les éléments techniques sous-jacents, en expliquant les choix faits à chaque étape pour optimiser les performances et la sécurité du système.

L'ensemble des 6 différentes parties qui expliquent le fonctionnement du code d'interopérabilité seront expliquées respectivement de la manière suivante :
La première partie expose le processus de réception des données provenant de chaque équipement par la station, ainsi que leur traitement. Un diagramme global illustre les différentes composantes abordées, identifiées de a à f. Chaque composante est ensuite analysée en détail dans les sous-sections qui lui sont spécifiquement consacrées.
La deuxième partie présente le raisonnement sous-jacent au chiffrement réalisé. Trois étapes distinctes sont décrites, chacune illustrant un aspect clé du processus. Un schéma accompagne l'explication et met en évidence l'enchaînement de ces étapes à l'aide d'une valeur fictive, afin de faciliter la compréhension.
La troisième partie détaille le mécanisme de transfert des données, élément crucial pour garantir un fonctionnement fluide de la station. Le protocole employé est présenté et justifié en fonction des exigences techniques et sécuritaires. Par ailleurs, le logiciel *Tailscale* est introduit comme solution de mise en réseau. Son rôle dans le système, ainsi que les raisons ayant motivé son adoption par la station, sont expliqués de manière approfondie.

La quatrième partie aborde la réception des données et leur traitement, suivant une logique similaire à celle de la première partie. Un diagramme global illustre les différentes composantes considérées, identifiées de a à d, et chacune d'elles est analysée en détail dans les sous-sections qui lui sont spécifiquement consacrées.
La cinquième partie présente le raisonnement sous-jacent au déchiffrement effectué à partir du résultat du chiffrement. Trois étapes distinctes sont détaillées afin d'illustrer le cheminement vers le résultat attendu. Un schéma accompagne l'explication et représente chacune des étapes, en utilisant la même valeur fictive que celle employée lors du chiffrement, afin d'assurer la cohérence de l'ensemble. Puis, la sixième partie expose la méthode pensée pour rendre la station totalement autonome.

Enfin, il sera mis en lumière le transfert des connaissances liées au code développé, assurant une compréhension et une pérennité de son utilisation. Elle est suivie de la présentation du diagramme de Gantt et des indicateurs clés de performance, permettant d'évaluer l'efficacité de la gestion de projet adoptée durant le stage. Une première conclusion apporte une vue d'ensemble sur l'organisation et la répartition des tâches confiées, mettant en évidence les points forts du processus mis en place. Enfin, une seconde conclusion souligne l'aboutissement du développement du code, la pertinence des résultats obtenus, et propose une vision prospective quant à l'évolution future du projet.

2 Présentation de l'entreprise

L'Institut National INDICATIC AIP a été fondé le 4 octobre 2019 en tant qu'association d'intérêt public (AIP [1]) avec une personnalité juridique.

La mission de l'INDICATIC AIP est d'attirer des talents et de les orienter vers le développement technologique bénéfique pour le Panama, tout en contribuant au progrès technologique mondial. L'objectif principal est de positionner le Panama à la frontière de la science, de la technologie et de l'innovation dans le domaine des TIC [2].

Les objectifs spécifiques de l'institut incluent le recrutement de chercheurs de haut niveau, la création d'un environnement propice à la recherche et développement (*R&D*) de pointe, le développement de connaissances et de technologies avancées, et la contribution à un agenda national de *R&D*. De plus, l'INDICATIC AIP s'engage à former des ressources humaines de qualité.

Dans le cadre de son engagement à promouvoir l'innovation, l'INDICATIC réalise des appels à projets et propose également des projets dans le domaine de l'informatique. Ces initiatives visent à répondre aux défis technologiques actuels, en collaborant avec des partenaires nationaux et internationaux pour développer des solutions avancées. Le laboratoire travaille activement à attirer des financements et à proposer des projets de recherche dans des domaines clés des technologies de l'information et de la communication, tout en soutenant l'écosystème d'innovation technologique au Panama.

L'institut est situé sur la Vía Centenario, au Campus Víctor Levis Sasso de l'Université Technologique de Panama, dans l'Édifice des Laboratoires de Recherche et d'Innovation, au 3ème étage. L'organigramme de son conseil d'administration est représenté dans la figure 3.

Figure 1 – *Logo de INDICATIC*

Figure 2 – *Édifice des Laboratoires de Recherche et d'Innovation*

1. Association d'Intérêt Public.
2. Technologies de l'Information et de la Communication.

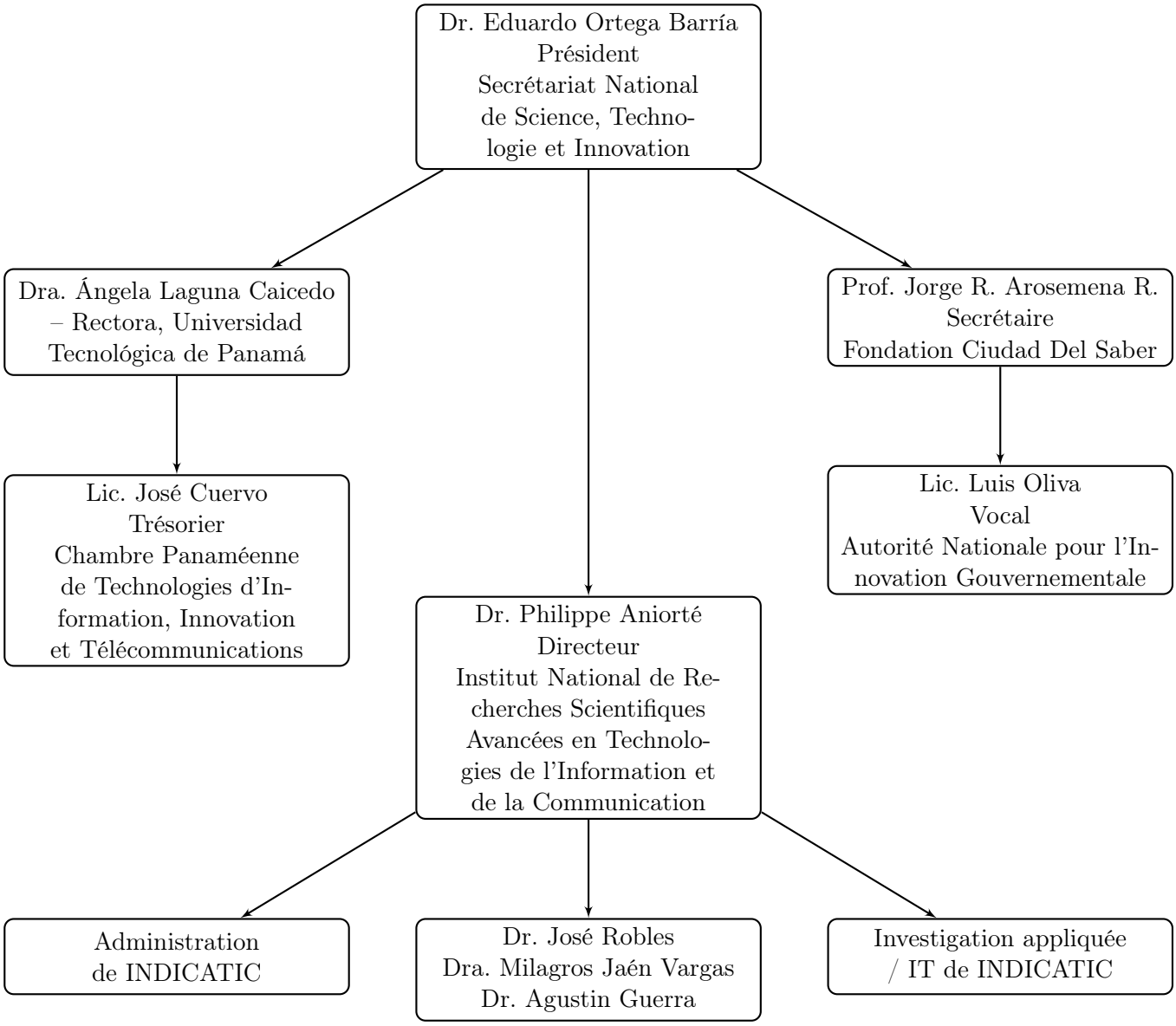

Figure 3 – *Organigramme du Conseil d'Administration et des effectifs de INDICATIC AIP*

3 Contexte

Le code d'interopérabilité permet l'envoi autonome et fluide de données issues de formats hétérogènes (`FIT`, `RAW`, `CSV`, `NEF`). Lors de leur réception, l'intégrité des données est systématiquement vérifiée à l'aide d'un procédé cryptographique adapté à chaque station émettrice, comme détaillé dans les sections 9 et 12. Ce mécanisme garantit la fiabilité des échanges et la sécurité des données transmises.
Le code doit également s'adapter dynamiquement à la station dans laquelle il est déployé. Cela signifie que les données envoyées doivent être correctement dirigées vers le répertoire approprié, en fonction de la station d'origine. Cette interopérabilité est essentielle pour assurer la continuité du service automatique de chaque station.
En cas de perte de connexion Internet, de défaillance des appareils connectés à la station ou de surcharge de la mémoire de stockage, le code est conçu pour réagir intelligemment : certains processus sont temporairement arrêtés, puis relancés automatiquement dès que les conditions redeviennent favorables. Ce comportement garantit une robustesse opérationnelle même dans des environnements instables.
Le climat chaud du Panama, combiné à la charge de travail élevée des stations, peut entraîner des dysfonctionnements matériels. Bien que les équipements soient compatibles avec Windows, ce système est jugé trop lourd et inadapté aux objectifs du projet, qui forcerait à réaliser les codes avec WSL (Windows Subsystem for Linux) [3]. L'approche retenue repose sur l'utilisation de **INDI (Instrument Neutral Distributed Interface)**, présentée dans la section 7, qui permet de contrôler les équipements directement via du code `C++`. Cette solution permet l'installation d'un système Linux sur chaque station, réduisant les tâches en arrière-plan et contribuant à limiter la température interne, ce qui diminue les risques de panne.
Le traitement des données est réalisé en `Shell`, pour sa légèreté et son efficacité, tandis que la partie cryptographique est développée en `C`, permettant une gestion fine de la mémoire et un code optimisé permettant le renforcement des données par des mécanismes de chiffrement spécifiques à chaque station. Les accès aux répertoires sont contrôlés par des permissions strictes, et une journalisation des transferts est mise en place pour assurer la traçabilité et faciliter les audits.
Chaque station doit être capable de se contrôler entièrement de manière autonome et d'envoyer environ **120 Go de données par nuit**. Ces données sont ensuite traitées et organisées dans le système de stockage pour être finalisées au cours de la journée suivante.
Le code est publié sur GitHub afin de faciliter sa compréhension, son installation et son adaptation à d'autres stations. Il est conçu pour être repris facilement par d'autres développeurs, permettant des ajouts ou des modifications sans complexité excessive. Des tests automatisés, incluant des tests unitaires, de charge et de robustesse, sont réalisés pour garantir la fiabilité du système. Des simulations de panne sont également effectuées pour valider les mécanismes de reprise.
Dans une perspective de déploiement à plus grande échelle, l'utilisation de **Tailscale**, détaillée dans la subsection 10.2, permet une communication directe entre les stations. Cela ouvre la voie à des envois synchronisés, qui permettraient l'optimisation de la gestion de la mémoire du dispositif de réception et renforcent la cohérence du réseau.

3. fonctionnalité de Windows qui permet d'exécuter un environnement Linux directement sur Windows, sans avoir besoin d'une machine virtuelle ou d'un double démarrage.

4 Analyse de l'existant

En février, le projet en était encore à un état préliminaire. Les équipements de mesure étaient déjà disponibles, mais aucun code ne permettait encore leur fonctionnement automatique. Jusqu'à cette étape, il était nécessaire d'utiliser des applications spécifiques à chaque appareil pour pouvoir capturer des données.
Les équipements impliqués sont les suivants : **Jetson Orin Nano (station centrale), Alpy 600, QHY16200A, Nikon D5600a, TESS-W, TESS-4C et un NAS Synology DS1821+**.
Le Jetson Orin Nano joue le rôle de station : il centralise la connexion des différents instruments et assure la transmission des données vers le système de stockage. Les caméras et spectromètres y sont reliés directement par câble USB, tandis que les photomètres TESS-W et TESS-4C sont connectés en Wi-Fi.
Le **NAS Synology DS1821+** assure la gestion et le stockage des données collectées. Il permet également un accès distant sécurisé et l'intégration d'outils complémentaires tels que Tailscale, ce qui facilite la mise en réseau des différentes stations. Ainsi, l'architecture actuelle repose sur une station centrale qui regroupe l'ensemble des équipements de mesure et un NAS qui centralise et sécurise le stockage des données.

5 Cahier des charges

En prenant en compte les contraintes techniques du projet — telles que la diversité des formats de fichiers (FIT, RAW, CSV, NEF), la nécessité d'un traitement autonome et sécurisé, la robustesse face aux défaillances matérielles, et l'adaptabilité à chaque station — une architecture spécifique a été pensée pour aboutir à un système d'interopérabilité fiable et évolutif. Ce système repose sur six parties principales, chacune jouant un rôle déterminant dans le traitement, la sécurisation et le transfert des données entre les stations et le système de stockage situé dans les locaux de INDICATIC AIP.
La **Partie 1** est dédiée au traitement des données côté station. Elle représente le code d'interopérabilité qui gère la préparation et l'envoi des informations vers le NAS, en tenant compte du répertoire cible propre à chaque station. La **Partie 2** se concentre sur la cryptographie, plus précisément sur le chiffrement. Elle décrit la méthode utilisée pour générer les empreintes numériques des fichiers *Payload* et *info*, qui sont ensuite enregistrées dans le fichier *CryptFile.txt*. Ces empreintes permettront au NAS d'identifier les fichiers envoyés par chaque station et de garantir leur intégrité. La **Partie 3** traite du transfert des données. Elle couvre le mécanisme par lequel les fichiers préparés et chiffrés sont transmis de la station vers le NAS, en assurant une fluidité même en cas de surcharge ou de perte de connexion temporaire. Une fois les données reçues, la **Partie 4** prend le relais pour effectuer leur traitement côté NAS. Cette étape consiste à analyser, organiser et stocker les fichiers reçus dans le système de stockage prévu à cet effet. La **Partie 5** revient à la cryptographie, mais cette fois du point de vue du déchiffrement. Le NAS utilise le fichier *CryptFile.txt* pour retrouver les empreintes numériques des fichiers reçus et les comparer avec celles qu'il calcule lui-même, afin de vérifier leur intégrité et leur authenticité. Enfin, la **Partie 6** concerne l'interface de contrôle. Elle permet de lancer automatiquement les trois premières parties (traitement, chiffrement et transfert) sur une station, assurant ainsi le bon fonctionnement du processus d'interopérabilité avec la couche cryptographique, tout en s'adaptant dynamiquement à l'environnement matériel et logiciel de chaque station.
Également, il devait être possible de communiquer avec les appareils, Alpy, QHY et Nikon directement avec des scripts.

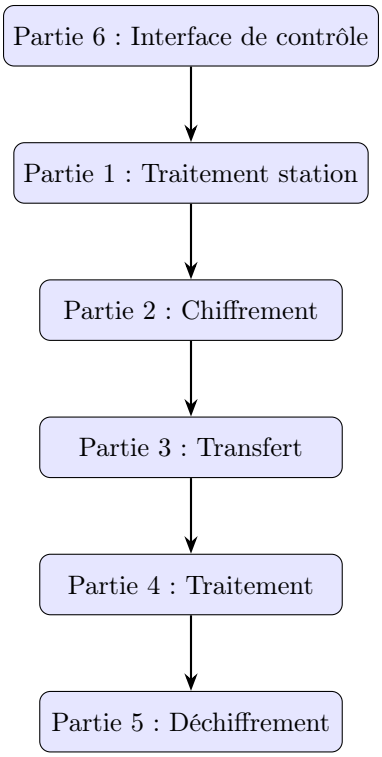

Figure 4 – *Flux des traitements entre station et NAS*

5.1 Vue d'ensemble

Sur la figure 5 ci-dessous figure le schéma qui représente le code d'interopérabilité conceptualisé. Sur le schéma sont représentées les différentes parties de 1 à 5. La partie 6 n'est pas représentée car celle-ci est à part. Le fonctionnement du code de la partie 6 est présenté plus tard dans le rapport plus en détail. Chaque partie est également expliquée plus en détail dans les différentes sections du rapport qui leur sont accordées.

Afin d'avoir une compréhension plus facile du schéma, voici quelques explications. La partie 1 représentée en rouge sur le côté gauche représente les données capturées par les équipements. Par exemple, pour le cas d'Alpy, le code va prendre les données d'Alpy s'il y en a, et va passer à QHYCCD ensuite et ainsi de suite avec les autres équipements indéfiniment pendant le fonctionnement du code. Une fois les données récupérées d'un équipement, les empreintes numériques des fichiers *Payload* et *info* (fichier *info* qui contient les métadonnées) sont générées, puis la partie 2 (cryptographique (chiffrement)) est exécutée avant d'envoyer les données vers le NAS via le protocole SFTP [4](Secure File Transfer Protocol).

Une fois que les données sont donc envoyées vers le NAS (partie 3), la partie 4 représente ce qu'il se passe du côté du NAS, celui-ci regarde en boucle si des fichiers sont reçus dans les différents répertoires (les différents chemins de chaque station). Si des fichiers sont reçus, alors est exécutée la génération de leur empreinte numérique de la même manière que dans les stations avec le même algorithme (sha256 [5]) puis la cryptographie de déchiffrement (partie 5) est effectuée sur les fichiers *Payload* et *info* reçus afin de comparer les empreintes de chacun. Si elles correspondent, alors le NAS retourne à chercher des fichiers à partir du chemin suivant. Si les empreintes ne correspondent pas, alors les fichiers concernés sont supprimés et le NAS poursuit sa recherche de fichiers à traiter à partir du chemin suivant également.

4. protocole réseau sécurisé utilisé pour le transfert de fichiers.
5. algorithme de hachage cryptographique qui génère un hachage de 256 bits (32 octets) à partir de n'importe quelle donnée d'entrée.

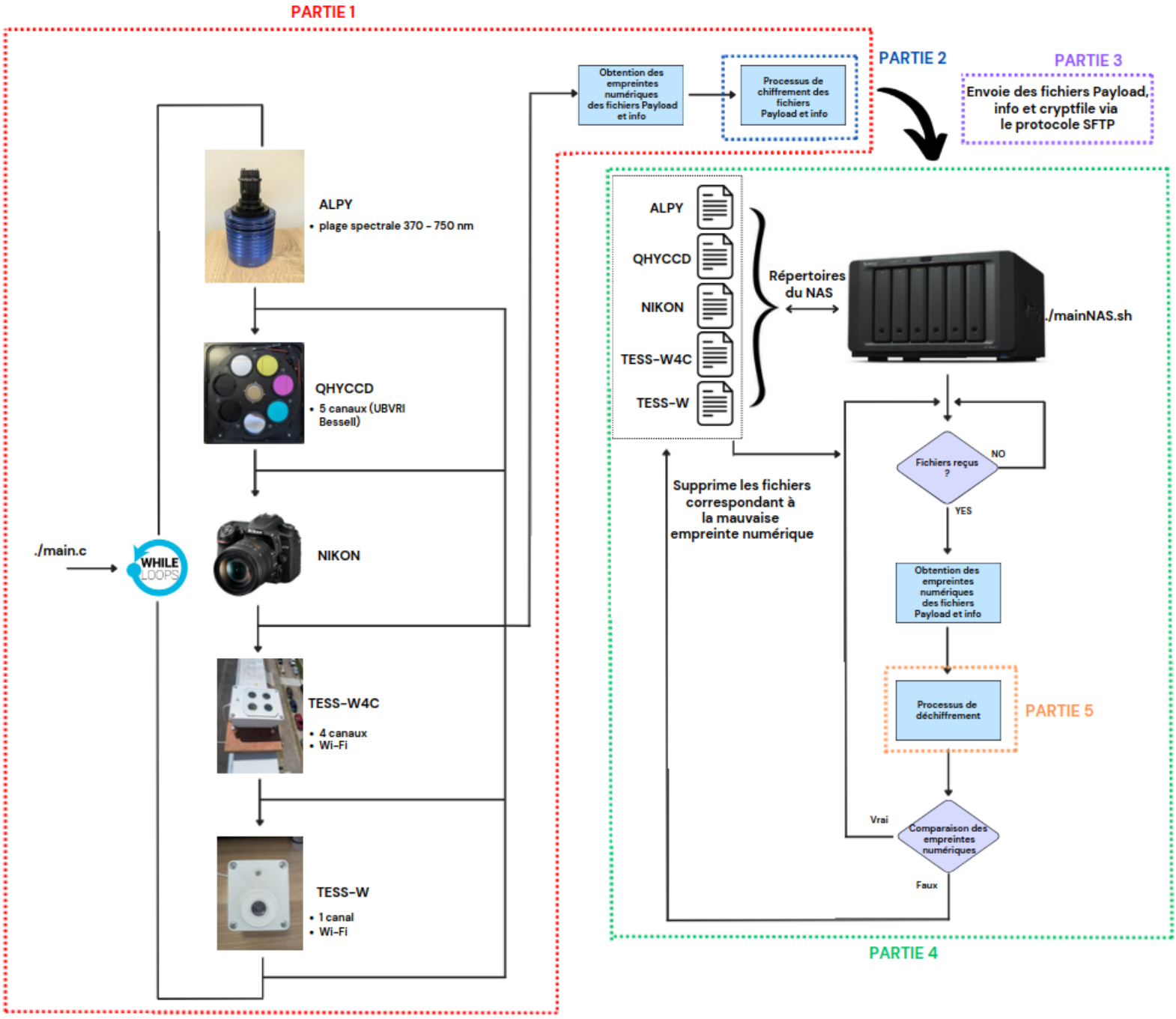

Figure 5 – *Vue d'ensemble du fonctionnement du code d'interopérabilité*

6 INDI

Comme indiqué précédemment, la consommation importante de ressources propre à Windows, l'utilisation des équipements Alpy et QHY via ce système pour la gestion des prises de données ne permet pas une communication directe avec les appareils à travers des langages informatiques. Cela restreint considérablement les possibilités de personnalisation et d'automatisation, notamment pour des opérations telles que : La question de l'utilisation d'une application soulève plusieurs limites. D'abord, en termes de **praticité**, le fait de ne pas pouvoir interagir directement avec un code empêche les modifications ainsi que le contrôle à distance. Ensuite, concernant la **possibilité**, une application ne fournit pas nécessairement les métadonnées indispensables relatives aux différentes prises de données. Enfin, sur le plan de la **disponibilité**, elle ne permet pas le contrôle à distance et complique toute intervention en cas de défaillance.

Afin de pallier à cela, il y a INDI qui est un protocole open source et une bibliothèque logicielle conçus pour permettre à des logiciels de contrôler une grande variété d'équipements astronomiques. Dans ces équipements figurent les modèles d'Alpy et QHY utilisés pour le projet. Il a donc été repris les codes de GitHub de INDI qui permettent de communiquer avec ces équipements afin de les personnaliser au besoin souhaité pour le projet.

6.1 Fonctionnement

INDI repose sur une architecture client-serveur, le serveur INDI (indiserver) gère la communication entre les pilotes matériels (drivers) et les logiciels clients. Dans le cadre du projet, le client utilisé est libindi, qui est la bibliothèque cliente officielle du protocole INDI en C++ afin de pouvoir se connecter à indiserver, interagir avec les périphériques et envoyer/recevoir des propriétés INDI sous forme XML (eXtensible Markup Language [6]) via des objets C++.

Afin d'avoir un meilleur aperçu des différents rôles de chaque élément, une représentation structurée des différents rôles des éléments mentionnés est présentée dans le tableau 1.

Table 1 – *Rôles des composants dans l'architecture INDI*

Élément	**Rôle**
`indiserver`	Serveur central INDI
QHY, Alpy	Périphériques gérés par pilotes INDI
Code C++	Client INDI via `libindi`

Ci-dessous les figures 6 et 7 présentent respectivement les commandes *indiserver* et *indi_getprop* dont les fonctions sont, d'une part, de lancer le serveur INDI avec le chargement d'un pilote spécifique (ici celui de la caméra QHYCCD (Charged-Coupled Device) [7]), et d'autre part, d'interroger un serveur INDI afin d'obtenir la valeur des propriétés exposées par les appareils connectés.

Sur la figure 6 est visualisable que le serveur INDI démarre et ouvre le port 7624 pour permettre aux clients INDI de se connecter et crée également un socket local Unix pour les connexions internes. Le pilote est lancé dans un processus enfant qui a pour PID (Process IDentifier) 39698. Par la suite, plusieurs drivers sont affichés, où chaque driver INDI assure un rôle spécifique à un matériel donné et expose les fonctions que ce matériel est capable de réaliser, un exemple est démontré sur le tableau 2, la ligne *Client 9 : new arrival from 127.0.0.1 :56850 - welcome !* indique qu'un nouveau client vient de se connecter au serveur INDI en connexion locale (l'équipement QHYCCD dans ce cas), *Client 9 :*

6. Pour le cas du projet cela représente les propriétés exposées par les périphériques (température, temps de mesure, connexion...).
7. Type de capteur utilisé pour capturer des images, très courant en astrophotographie, microscopie, caméras scientifiques, etc.

read EOF (End Of File) signifie par la suite que le client a fermé la connexion, enfin *Client 9 : shut down complete - bye !* le serveur termine proprement la session avec ce client.

Table 2 – *Correspondance entre les types de demande et les propriétés INDI*

Type de demande	Propriété INDI exposée par le driver
Définir une durée de pose	`CCD_EXPOSURE`
Lire la température du capteur	`CCD_TEMPERATURE`
Régler le gain	`CCD_GAIN`
Sélectionner une région ROI (Region of Interest)	`CCD_FRAME`
Déclencher une capture	`CCD_CAPTURE` ou `CCD_EXPOSURE`
Recevoir une image	`CCD1` (type BLOB – données image brutes binaire)

```
(base) alex@Gam:~$ indiserver -v indi_qhy_ccd
2025-07-07T20:08:01: startup: indiserver -v indi_qhy_ccd
2025-07-07T20:08:01: Driver indi_qhy_ccd: pid=39698 rfd=6 wfd=6 efd=7
2025-07-07T20:08:01: listening to port 7624 on fd 5
2025-07-07T20:08:01: Local server: listening on local domain at: @/tmp/indiserver
2025-07-07T20:08:01: Driver indi_qhy_ccd: -- qhyccd.cpp param
2025-07-07T20:08:01: Driver indi_qhy_ccd: QHYCCD|QHYCCD.CPP|InitQHYCCDResource()|START
2025-07-07T20:08:01: Driver indi_qhy_ccd: QHYCCD|QHYCCD.CPP|InitQHYCCDResource|auto_detect_camera:false,call InitQHYCCDResourceInside
2025-07-07T20:08:01: Driver indi_qhy_ccd: QHYCCD|QHYCCD.CPP|InitQHYCCDResourceInside|START
2025-07-07T20:08:01: Driver indi_qhy_ccd: QHYCCD|QHYCCD.CPP|libusb_version 1.0.27.11882
2025-07-07T20:08:01: Driver indi_qhy_ccd: QHYCCD|QHYCCD.CPP|libusb_init(libqhyccd_context) called...
2025-07-07T20:08:01: Driver indi_qhy_ccd: QHYCCD|QHYCCD.CPP|InitQHYCCDResourceInside|numdev set to 0
2025-07-07T20:08:01: Driver indi_qhy_ccd: QHYCCD|QHYCCD.CPP|InitQHYCCDResourceInside|END
2025-07-07T20:08:01: Driver indi_qhy_ccd: ************************ config file path  25.6.3.12 svn: 1  **********************************
2025-07-07T20:08:01: Driver indi_qhy_ccd: QHYCCD|QHYCCD.CPP|InitQHYCCDResource|Load ini filePath = /home/alex  fileName = qhyccd.ini
2025-07-07T20:08:02: Driver indi_qhy_ccd: snooping on Telescope Simulator.EQUATORIAL_EOD_COORD
2025-07-07T20:08:02: Driver indi_qhy_ccd: snooping on Telescope Simulator.EQUATORIAL_COORD
2025-07-07T20:08:02: Driver indi_qhy_ccd: snooping on Telescope Simulator.TELESCOPE_INFO
2025-07-07T20:08:02: Driver indi_qhy_ccd: snooping on Telescope Simulator.GEOGRAPHIC_COORD
2025-07-07T20:08:02: Driver indi_qhy_ccd: snooping on Telescope Simulator.TELESCOPE_PIER_SIDE
2025-07-07T20:08:02: Driver indi_qhy_ccd: snooping on Rotator Simulator.ABS_ROTATOR_ANGLE
2025-07-07T20:08:02: Driver indi_qhy_ccd: snooping on Focuser Simulator.ABS_FOCUS_POSITION
2025-07-07T20:08:02: Driver indi_qhy_ccd: snooping on Focuser Simulator.FOCUS_TEMPERATURE
2025-07-07T20:08:02: Driver indi_qhy_ccd: snooping on CCD Simulator.FILTER_SLOT
2025-07-07T20:08:02: Driver indi_qhy_ccd: snooping on CCD Simulator.FILTER_NAME
2025-07-07T20:08:02: Driver indi_qhy_ccd: snooping on SQM.SKY_QUALITY
2025-07-07T20:13:03: Client 9: new arrival from 127.0.0.1:56850 - welcome!
2025-07-07T20:13:05: Client 9: read EOF
2025-07-07T20:13:05: Client 9: shut down complete - bye!
```

Figure 6 – *Résultat commande indiserver*

```
(base) alex@Gam:~$ indi_getprop
QHY CCD 16200A-M-0047e4.CONNECTION.CONNECT=Off
QHY CCD 16200A-M-0047e4.CONNECTION.DISCONNECT=On
QHY CCD 16200A-M-0047e4.DRIVER_INFO.DRIVER_NAME=QHY CCD
QHY CCD 16200A-M-0047e4.DRIVER_INFO.DRIVER_EXEC=indi_qhy_ccd
QHY CCD 16200A-M-0047e4.DRIVER_INFO.DRIVER_VERSION=2.8
QHY CCD 16200A-M-0047e4.DRIVER_INFO.DRIVER_INTERFACE=2
QHY CCD 16200A-M-0047e4.POLLING_PERIOD.PERIOD_MS=1000
QHY CCD 16200A-M-0047e4.DEBUG.ENABLE=Off
QHY CCD 16200A-M-0047e4.DEBUG.DISABLE=On
QHY CCD 16200A-M-0047e4.SIMULATION.ENABLE=Off
QHY CCD 16200A-M-0047e4.SIMULATION.DISABLE=On
QHY CCD 16200A-M-0047e4.CONFIG_PROCESS.CONFIG_LOAD=Off
QHY CCD 16200A-M-0047e4.CONFIG_PROCESS.CONFIG_SAVE=Off
QHY CCD 16200A-M-0047e4.CONFIG_PROCESS.CONFIG_DEFAULT=Off
QHY CCD 16200A-M-0047e4.CONFIG_PROCESS.CONFIG_PURGE=Off
QHY CCD 16200A-M-0047e4.ACTIVE_DEVICES.ACTIVE_TELESCOPE=Telescope Simulator
QHY CCD 16200A-M-0047e4.ACTIVE_DEVICES.ACTIVE_ROTATOR=Rotator Simulator
QHY CCD 16200A-M-0047e4.ACTIVE_DEVICES.ACTIVE_FOCUSER=Focuser Simulator
QHY CCD 16200A-M-0047e4.ACTIVE_DEVICES.ACTIVE_FILTER=CCD Simulator
QHY CCD 16200A-M-0047e4.ACTIVE_DEVICES.ACTIVE_SKYQUALITY=SQM
QHY CCD 16200A-M-0047e4.ACTIVE_DEVICES.ACTIVE_TELESCOPE=Telescope Simulator
QHY CCD 16200A-M-0047e4.ACTIVE_DEVICES.ACTIVE_ROTATOR=Rotator Simulator
QHY CCD 16200A-M-0047e4.ACTIVE_DEVICES.ACTIVE_FOCUSER=Focuser Simulator
QHY CCD 16200A-M-0047e4.ACTIVE_DEVICES.ACTIVE_FILTER=CCD Simulator
QHY CCD 16200A-M-0047e4.ACTIVE_DEVICES.ACTIVE_SKYQUALITY=SQM
```

Figure 7 – *Résultat commande indi_ getprop*

Sur la figure 7 sont démontrées toutes les propriétés exposées par la caméra QHY CCD 16200A-M-0047e4, sur les deux premières lignes il est possible de voir que la caméra n'est pas actuellement connectée (CONNECT=Off), il est possible de constater le nom commercial du pilote (DRIVER_NAME), le nom du fichier exécutable du driver (DRIVER_EXEC), la version du pilote (DRIVER_VERSION) etc... Tous ces paramètres exposés par le pilote INDI sont entièrement accessibles et configurables depuis le code C++, grâce à l'API de libindi utilisée.

7 Partie 1 : De l'éxécution du code jusqu'au moment d'avant l'envoie

Dans cette première partie qui représente la partie 1 représentée sur la figure 5, il est expliqué le fonctionnement du code d'interopérabilité de son exécution jusqu'à avant l'envoi des données, excepté la partie 2 (cryptographie (chiffrement)) qui sera expliquée dans la partie suivante 8. Il est dans un premier temps expliqué le diagramme général de la partie 1 dans la section 7.1 puis par la suite chaque partie du diagramme de *a* à *f* sera expliquée et détaillée plus en détail.

7.1 Diagramme général et explication

Le diagramme général de la partie 1 est représenté sur la figure 8, où les différents scripts représentés sont exécutés, les scripts ont été codés en langages C et bash dans le but d'avoir une rapidité d'exécution des codes la plus rapide possible. Cela peut être possible grâce au langage de bas niveau qui est C, et bash avec sa syntaxe étendue qui rend le codage plus simple.

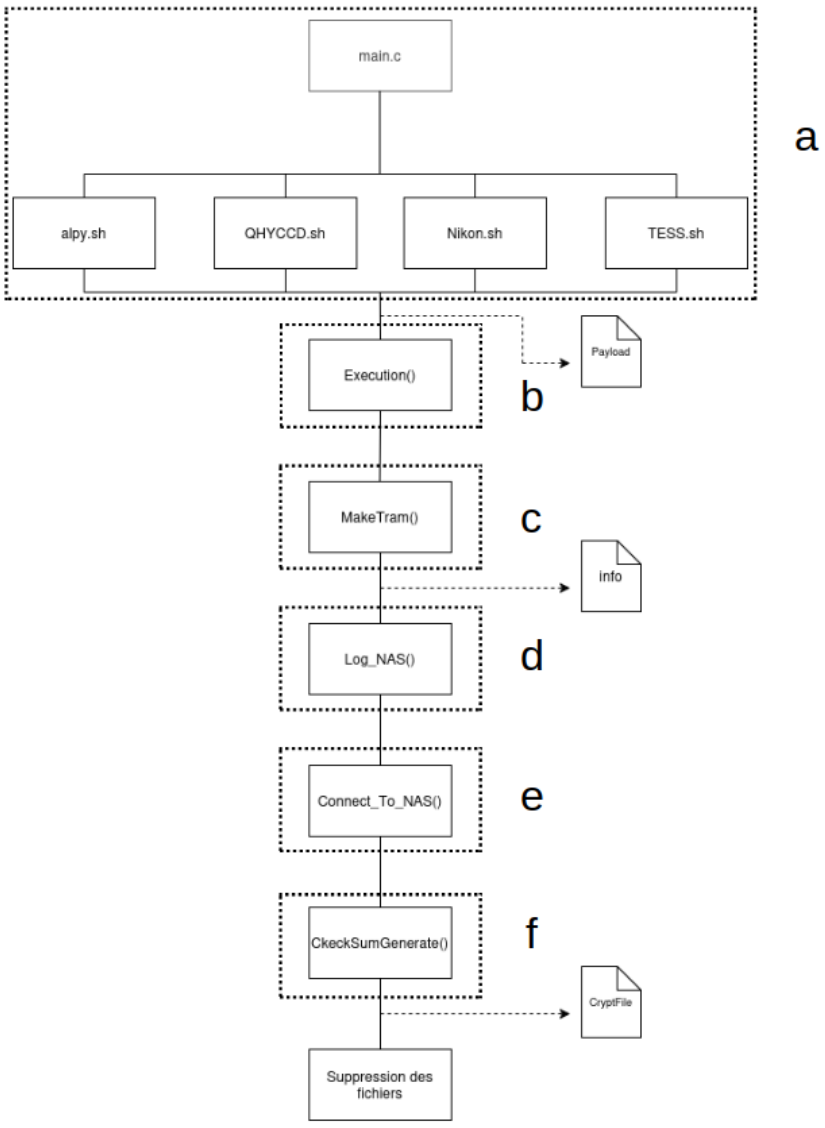

Figure 8 – *Diagramme Partie 1*

Le code commence par le script *main.c*, c'est à partir de ce script qu'est vérifié si des fichiers sont présents dans les répertoires où sont déposés les fichiers des équipements Alpy, QHY, Nikon, TESS-W et TESS-W4. Il est à noter que seul un dossier TESS-W est présent sur le diagramme, c'est parce que les fichiers de TESS-W et TESS-W4 sont envoyés dans le même répertoire.

Quand un fichier doit être traité, le code qui correspond au répertoire est donc exécuté, par exemple, si un fichier est présent dans le répertoire d' Alpy, alors le script *alpy.sh* sera exécuté, les variables ont les valeurs spécifiques pour un fichier Alpy attribué, puis la fonction *Execution* est appelée, cette fonction contient tout le traitement jusqu'à la fin, c'est-à-dire qu'elle s'occupe du traitement des données, de l'envoi, et de leur suppression sur la station une fois les données envoyées. Suite à cela, est appelée la fonction *MakeTram*, c'est dans cette partie du code où sont ajoutées les métadonnées pour être envoyées dans le fichier *info* qui est visualisable sur le diagramme. La fonction *Log_ NAS* qui suit fournit les informations de connexion pour se connecter au compte INDICATIC sur le NAS avec le chemin auquel doivent être envoyés les fichiers,

l'avant-dernière fonction *Connect_ To_ NAS* appelle la fonction *CheckSumGenerate*, puis gère plusieurs vérifications auprès du NAS comme les doublons et gère le cas s'ils sont présents, puis envoie les fichiers vers le NAS. Quant à elle, la fonction *CheckSumGenerate* génère les empreintes numériques pour les fichiers *Payload* et *info* puis exécute le script principal chargé de réaliser le chiffrement.

7.2 Exécution de main.c et traitement des données de chaque équipement (a)

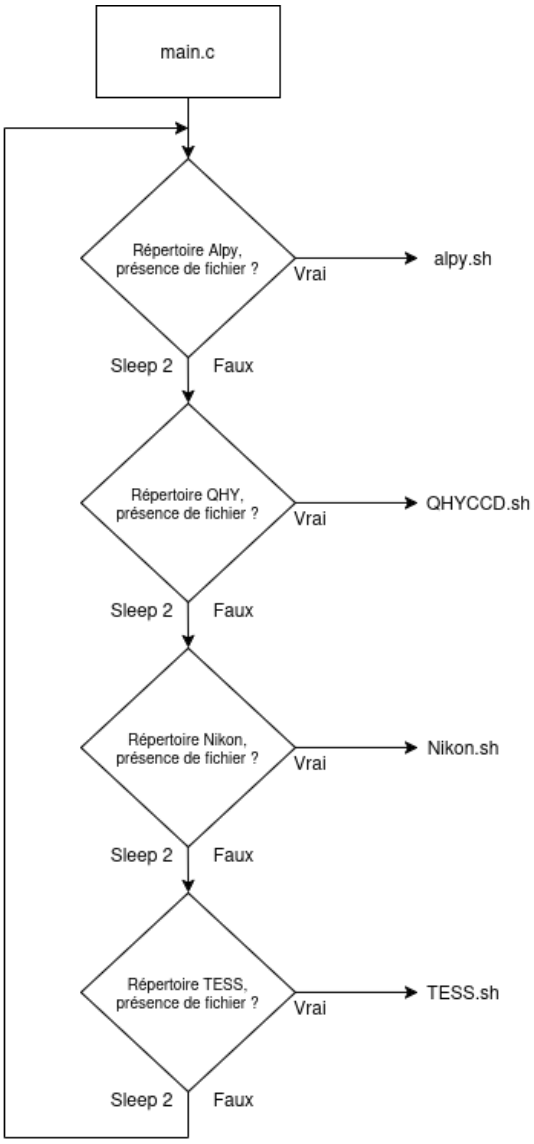

Figure 9 – *Diagramme partie a*

Sur la figure 9 est représentée la partie *a* du schéma de la partie 1. Sur ce schéma, il est possible de voir en détail comment les fichiers des différents appareils de mesures, Alpy, QHY, Nikon, TESS-W et TESS-W4 sont traités afin de déterminer si au moins un fichier est présent dans leur répertoire respectif. Une fois que le script *main.c* est exécuté, le code entre donc dans une boucle infinie, à chaque 2 secondes d'intervalle le code vérifie si au moins un fichier est détecté dans un répertoire puis passe au suivant, si plusieurs fichiers sont présents alors le fichier le plus ancien est sélectionné et le script respectif associé au dossier du fichier est exécuté.

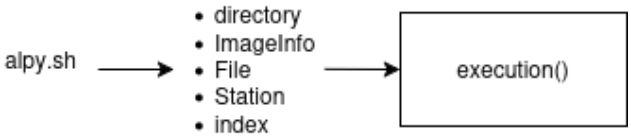

Figure 10 – *Déroulement du script alpy.sh*

La figure 10 montre le déroulement du script *alpy.sh*, sachant que les quatre fichiers .sh du diagramme de gauche ont le même déroulement, seulement l'explication de alpy peut être faite, une fois que le fichier *alpy.sh* est exécuté, les variables *directory, imageInfo, File, Station et index* vont respectivement avoir les valeurs suivantes attribuées : le chemin vers le dossier où se situe le fichier *Payload*, le chemin vers le répertoire où sera le fichier *info* associé au fichier *Payload*, la variable qui contient le *Payload*, le chemin de la station dans le NAS vers lequel les fichiers doivent être envoyés, la valeur qui correspond au dossier Alpy.

Afin de pouvoir avoir un aperçu visuel du code et ainsi avoir une meilleure compréhension, la figure 11 montre comment les variables citées qui sont visibles sur la figure 10 ont leurs valeurs attribuées. Puis, pour terminer cette partie *a*, la fonction *execution* est exécutée par la suite.

```
Station=""
directory="/home/indicatic-e1/Desktop/ASTRODEVICES/ALPYFILE"
imageInfo="/home/indicatic-e1/Desktop/code/infoIMG"
File=$(ls -t --reverse "$directory" | head -n 1)

read -r Station < "/home/indicatic-e1/Desktop/code/NBstation.txt" #Put the name of the station into Station variable
echo "Path of the station is : $Station"

index=0

execution
```

Figure 11 – *Extrait du script alpy.sh concernant la figure 10*

7.3 Fonction execution, Séquenceur (b)

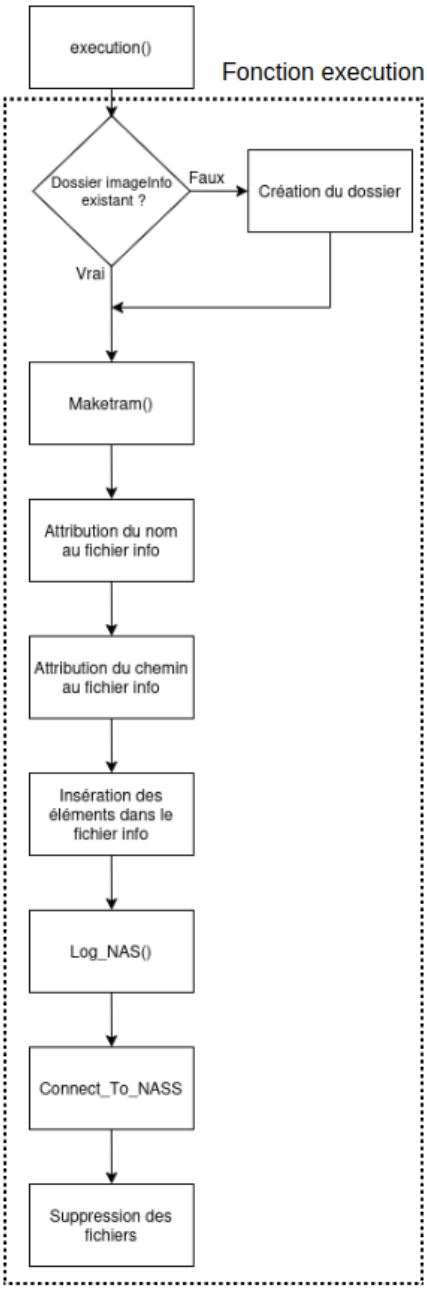

Figure 12 – *Diagramme Partie b*

La figure 12 représente la partie *b* du diagramme de la partie 1, en haut du diagramme est visualisable l'exécution de la fonction *execution* qui est représentée sur la figure 10, voici l'explication en détail de ce que fait donc cette fonction :

Premièrement, il est déterminé si le dossier *imageInfo* existe sur la station là où il devrait être. Si le fichier n'existe pas, alors il est créé et rejoint l'étape du cas s'il existait déjà. À partir de là, la fonction *Maketram* est exécutée et le nom du fichier *info* est attribué de la manière suivante :

```
newFileName="info_${FileName%.*}.txt"
```

la variable *FileName* est le chemin du *Payload*, donc avec l'option %.* Il est possible de supprimer l'expansion du *Payload*, il est ajouté *info_* avant le *Payload* et est également ajouté *.txt* à la fin et la variable *newFileName* contient maintenant le nom du fichier *info* associé à celui du *Payload*.

Afin que ce fichier *info* puisse avoir un chemin identifiable par la station pour pouvoir l'utiliser, la ligne de code ci-dessous lui attribue ainsi un chemin :

```
FileInfo="$imageInfo/$newFileName"
```

Comme il était démontré sur la figure 11, un chemin est affilié à la variable *imageInfo* qui pointe vers le répertoire */home/indicatic-e1/Desktop/code/infoIMG*, à celui-ci est ajouté la variable *newFileName* qui contient donc le nom du fichier *info*, puis afin d'envoyer les données dans ce fichier *info*, la ligne :

```
echo "$TrameToSend" > "$FileInfo"
```

permet d'insérer les métadonnées contenues dans la variable *TrameToSend* directement dans le fichier *info*.

Un exemple concret des fichiers *Payload* et *info* reçus est représenté sur la figure 13 avec des résultats directement pris du NAS concernant des fichiers de l'équipement QHY.

Il est donc possible de voir que le nom du premier fichier est :
Tue_May_27_15_30_27_2025_10000000us_CFW5.fits et donc pour définir le nom du fichier *info*, *info_* a été ajouté au début du nom et *.fits* a été remplacé par *.txt*. Suite à cela, les fonctions *Log_NAS* et *Connect_To_NAS* sont exécutées avant que les fichiers traités soient définitivement supprimés de la station.

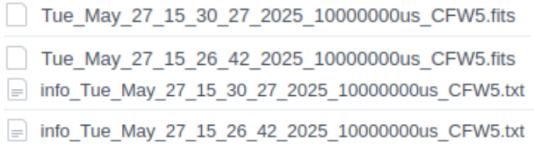

Figure 13 – *Exemple nommage fichier info*

7.4 Fonction Maketram, organisation des métadonnées (c)

```
MakeTram() {
  local directory="$1"

  #Assignment of the valors to their due variables
  #File=$(ls -1 "$directory" | head -n 1)
  FilePath="$directory/$File"
  FileName=$(basename "$FilePath")
  FileSize=$(stat --format=%s "$FilePath")
  LastModifTime=$(stat --format=%y "$FilePath")
  LastModifDay=$(date -r "$FilePath" +%A)
  RofAccess=$(stat --format=%A "$FilePath")
  FileFormat=$(file -b "$FilePath")

  #Tram architecture
  TrameToSend="$FileName $FileSize $LastModifTime $LastModifDay $RofAccess $FileFormat"
}
```

Figure 14 – *Script de la fonction Maketram*

Le script de la fonction *Maketram* est représenté sur la figure 14, le chemin du fichier *Payload* qui est *FilePath* permet ainsi de récupérer plusieurs métadonnées et de les attribuer à différentes variables afin que celles-ci soient concaténées à la variable *TrameToSend* afin d'être envoyées comme expliqué plus haut dans la partie 7.3.

7.5 Fonction Log_NAS, information de connexion (d)

```
Log_NASS() {
  #SFTP connexion information
  HOST="                "
  USER="          "
  PASSWD="                    "

  if [[ $index -eq 0 ]]; then
    REMOTE_DIR="/homes/INDICATIC/"$Station"/ALPY"
  elif [[ $index -eq 1 ]]; then
    REMOTE_DIR="/homes/INDICATIC/"$Station"/QHYCCD/U-Bessell"
  elif [[ $index -eq 2 ]]; then
    REMOTE_DIR="/homes/INDICATIC/"$Station"/QHYCCD/B-Bessell"
  elif [[ $index -eq 3 ]]; then
    REMOTE_DIR="/homes/INDICATIC/"$Station"/QHYCCD/V-Bessell"
  elif [[ $index -eq 4 ]]; then
    REMOTE_DIR="/homes/INDICATIC/"$Station"/QHYCCD/R-Bessell"
  elif [[ $index -eq 5 ]]; then
    REMOTE_DIR="/homes/INDICATIC/"$Station"/QHYCCD/I-Bessell"
  elif [[ $index -eq 6 ]]; then
    REMOTE_DIR="/homes/INDICATIC/"$Station"/NIKON"
  elif [[ $index -eq 7 ]]; then
    REMOTE_DIR="/homes/INDICATIC/"$Station"/TESS"$DESTINATION""
  else
    echo "error"
  fi

  LOCAL_FILE=$FilePath
}
```

Figure 15 – *Script de la fonction Log_NAS*

Le script de la fonction *Log_NAS* représenté sur la figure 15 permet d'avoir un aperçu global du traitement qui gère la destination vers le NAS de chaque appareil.

Premièrement, les informations de connexion nécessaires concernant le NAS tel que le nom de domaine "*HOST*", l'utilisateur ou les données vont être envoyées "*USER*", ainsi que le mot de passe "*PASSWD*" sont définies ici, et seront utilisées plus tard lors de l'envoi des données. Plus bas, est déterminé le chemin de destination dans le NAS de chaque appareil, la variable *Station* qui a sa valeur attribuée visualisable sur la figure 11 indique le chemin spécifique dans le NAS où doivent être envoyées les données (visualisables sur la figure 15), puis le chemin est complété avec le répertoire final de l'appareil qui est le même pour chaque station différente. Enfin, afin que cela soit le chemin de Alpy pour citer cet exemple, la valeur de la variable *index* détermine ainsi la valeur de la variable *REMOTE_DIR*, soit le chemin d'envoi.

7.6 Fonction Connect_To_NAS, transfert des données vers le NAS (e)

Cette fonction permet d'envoyer les données, soit les fichiers *Payload*, *info* et *CryptFile.txt* vers le NAS, également, une connexion au NAS est effectuée afin de detérminer si le NAS possède déjà des fichiers du même nom dans le répertoire d'envoi, la station traite donc de différentes manières l'envoi des données suivant le résultat. Enfin, une fois l'envoi effectué du côté de la station, celle-ci vérifie dans le NAS afin de confirmer si tous les fichiers envoyés ont bien été reçus.

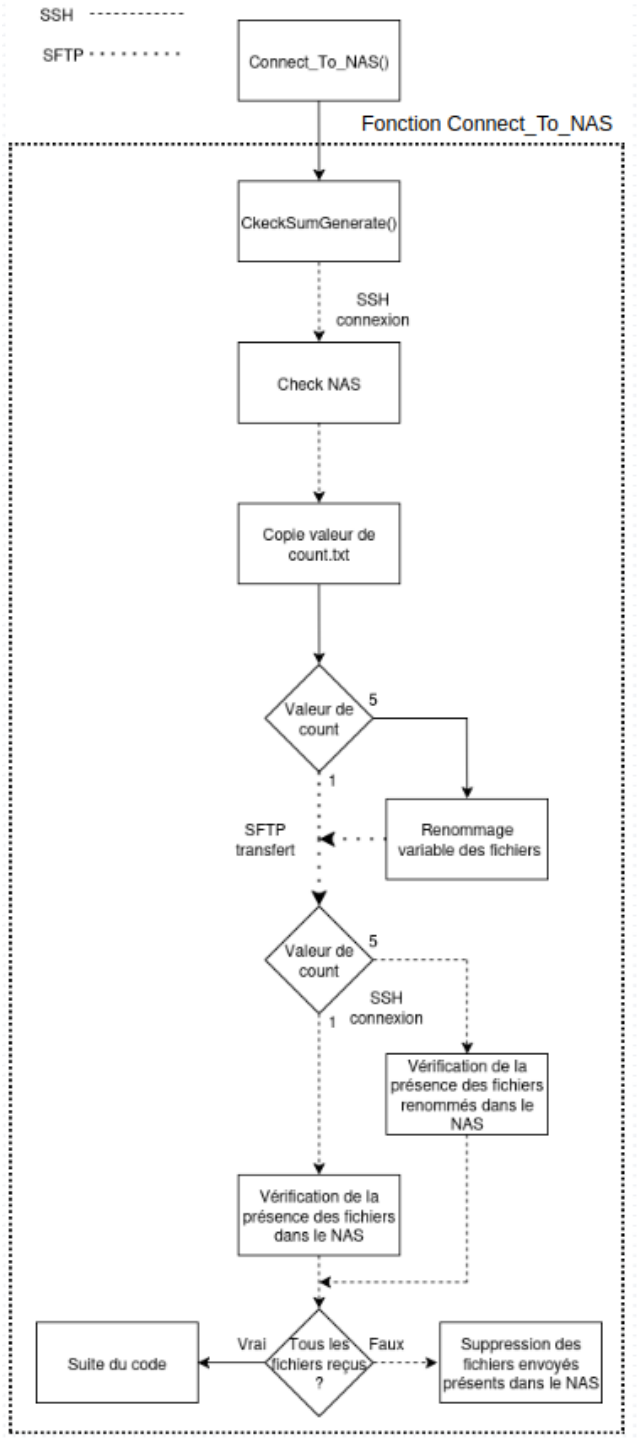

Figure 16 – *Diagramme Partie e*

La figure 16 représente la partie *e* du diagramme de la partie 1, c'est cette partie-là qui gère l'envoi des données vers le NAS ainsi que la vérification des doublons et de la réception complète des fichiers dans le NAS.

Premièrement, le script va exécuter la fonction *CheckSumGenerate*, cette fonction est celle qui sert à générer les empreintes numériques des fichiers *Payload* et *info*. Par la suite, une connexion SSH est établie vers le NAS pour rechercher si des fichiers présents dans le répertoire d'envoi ont le même nom que ceux à envoyer, si des fichiers ont le même nom, alors la valeur *5* est écrite dans le fichier *count.txt* présent dans le NAS ainsi qu'un chiffre qui correspond au numéro attribué pour le renommage, sinon c'est la valeur *1* qui est écrite pour signifier qu'aucun fichier ne possède le même nom, puis, cette valeur sera écrite dans le fichier *count.txt* de la station grâce à la commande :

```
sshpass -p "$PASSWD" scp "$USER@$HOST
```

suite à cela, avant l'envoi des fichiers, si la valeur est *5*, alors les fichiers sont renommés en ajoutant le chiffre écrit dans le fichier *count.txt* entre parenthèses (*-1* par défaut, ce chiffre a la valeur +1 par rapport au même nom des fichiers déjà traités) au début du nom des trois fichiers afin d'éviter que l'envoi des fichiers avec le même nom écrase les fichiers déjà présents dans le NAS.

À partir de là, les fichiers sont transférés vers le NAS via le protocole SFTP. Le choix de ce protocole est expliqué dans la partie 3. Une fois l'envoi des fichiers terminé, la station se connecte de nouveau en SSH afin de voir si les fichiers ont tous été reçus dans le NAS par rapport à la valeur écrite dans la variable *count* pour savoir quel fichier on recherche. Enfin, si tous les fichiers ont été reçus, alors la station passe à la suite du code ou alors, si un des fichiers manque, les fichiers qui correspondent sont supprimés du NAS.

Afin de mieux expliquer comment fonctionne l'organisation du fichier *count.txt*, le schéma pour le cas avec les valeurs *1* et *-1* (cas dans lequel aucun fichier existant dans le répertoire d'envoi du NAS n'a le même nom que les fichiers à envoyer) représenté sur la figure 17 explique plus en détail comment les valeurs sont insérées dans le fichier. La station réalise une connexion SSH et recherche des fichiers qui ont le même nom dans le répertoire de destination que les fichiers à envoyer, les valeurs sont donc insérées dans le fichier côté NAS puis avec la commande *scp* la station copie le fichier *count.txt* dans son environnement et attribue les deux valeurs du fichier dans des variables afin de procéder au traitement.

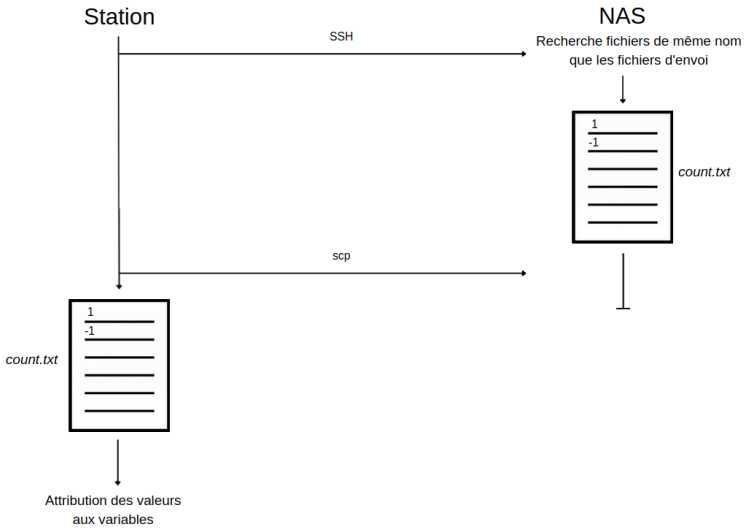

Figure 17 – *Organisation fichier count.txt, cas aucun fichier existant*

Sur la figure 18, un cas avec un fichier dans le NAS qui possède déjà le même nom que le fichier qui doit être envoyé est représenté. Donc, la première valeur est *5* pour signifier qu'un fichier est déjà existant, et le chiffre *1* est le chiffre incrémenté qui signifie que seulement 1 fichier du même nom est déjà présent, ce qui pourrait donner un exemple de nom pour les fichiers comme : *(1)nom du fichier*.

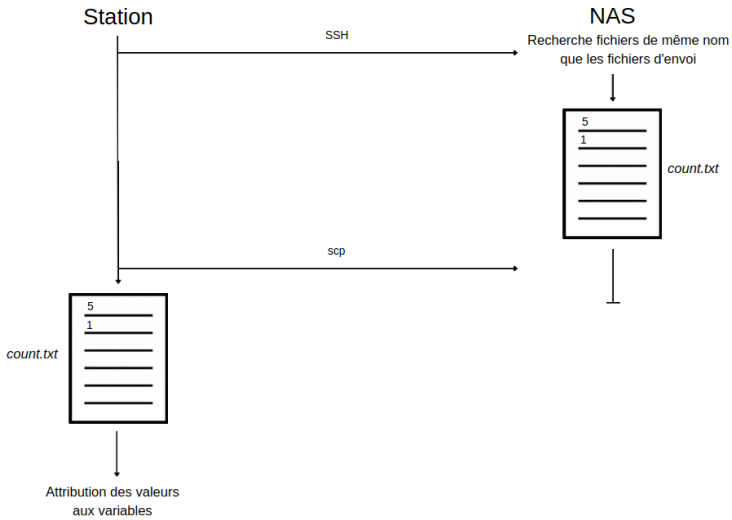

Figure 18 – *Organisation fichier count.txt, cas fichier existant*

7.7 Fonction ChecksumGenerate, réalisation des empreintes numériques (f)

```
CkeckSumGenerate() {
  #Checksum to be compared with what received the NAS to ensure the integrity of the payload
  checksumPayload=$(sha256sum "$LOCAL_FILE" | awk '{print $1}')
  echo "SHA-256 of the file : $checksumPayload"
  echo "local file : $LOCAL_FILE"

  #Checksum to be compared with what received the NAS to ensure the integrity of the info file
  checksumInfo=$(sha256sum "$FileInfo" | awk '{print $1}')
  echo "SHA-256 of the file info : $checksumInfo"

  #File in wich there is the checksum, this file have always the same name
  ChecksumToCrypt="$imageInfo"/"StringCrypt.txt"
  #File in which there is the checksum with the name of the appropriate file
  newFileNamechecksum="Checksum_${FileName%.*}.txt"

  ################################checksum treatment################################
  #Both checksum to crypt in the ChecksumToCrypt's file
  echo "$checksumPayload $checksumInfo" > "$ChecksumToCrypt"

  #Definition of the path
  FileInfochecksum="$imageInfo/$newFileNamechecksum"

  #Sending the crypted cheksum into the files
  echo -e "$checksumPayload\n$checksumInfo" > "$FileInfochecksum"

  #Execution of the encryption code
  EXECUTABLE="/home/indicatic-e1/Desktop/code/CryptageC/mainCrypt"
  $EXECUTABLE

}
```

Figure 19 – *ChecksumGenerate fonction*

La fonction *ChecksumGenerate* qui a pour objectif de générer les empreintes numériques des fichiers *Payload* et *info* est visualisable sur la figure 19, l'empreinte numérique du fichier *Payload* est générée avec la ligne du script suivante :

```
checksumPayload=$(sha256sum "$LOCAL_FILE" | awk '{print $1}')
```

puis la ligne :

```
checksumInfo=$(sha256sum "$FileInfo" | awk '{print $1}')
```

permet de générer l'empreinte numérique du fichier *info*. Suite à cela, il est donné à la variable *ChecksumToCrypt* le chemin vers le fichier *StringCrypt.txt* dans lequel sont insérées les empreintes numériques grâce à la ligne :

```
ChecksumToCrypt="$imageInfo"/"StringCrypt.txt"
```

les empreintes sont ensuite écrites dans le fichier qui est pointé par la variable *ChecksumToCrypt* avec la ligne :

```
echo "$checksumPayload $checksumInfo" > "$ChecksumToCrypt"
```

enfin, la partie cryptographique est exécutée pour procéder au chiffrement des deux empreintes numériques générées :

```
EXECUTABLE="/home/indicatic-e1/Desktop/code/CryptageC/mainCrypt"
$EXECUTABLE
```

8 Partie 2 : Chiffrement des empreintes numériques

Dans cette partie 2 sont expliquées les trois différentes étapes de chiffrement réalisées par le code d'interopérabilité sur les empreintes numériques de *Payload* et *info* dans le but de pouvoir transférer les informations nécessaires au NAS pour que celui-ci puisse les retrouver une fois reçues et les comparer, il est dans un premier temps expliqué le diagramme général de la partie 2 dans la section 8.1 puis par la suite chaque partie du diagramme de la première à la troisième étape sera expliquée et détaillée.

8.1 Diagramme général et explication

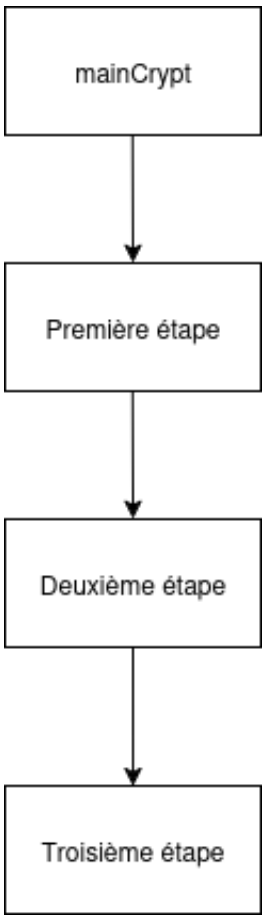

Figure 20 – *Diagramme Partie 2*

Sur la figure 20 sont représentées les trois différentes étapes de chiffrement lors de l'exécution de *mainCrypt*. La première étape consiste en le partage de clé secrète de Shamir (empreinte numérique dans ce cas) suivie de la deuxième étape qui entraîne un code de César sur le résultat de la première étape, enfin la troisième étape consiste en l'application d'un masque binaire sur le résultat de la deuxième étape. Une fois les trois étapes de chiffrement traitées, les données sont écrites dans le fichier *CryptFile.txt*.

8.2 Première étape : partage de clé secrète de Shamir

Dans cette première étape, l'objectif est d'obtenir n parts générées à partir du secret (payload ou info). Seules k parts (avec $k \leq n$) sont nécessaires pour reconstituer le secret dans la partie déchiffrement.

- Le secret est utilisé comme valeur du polynôme en $x = 0$: $f(0) secret$.
- Un polynôme aléatoire est construit avec un degré $k - 1$
- Des points $(x, f(x))$ sont générés sur ce polynôme

Dans le code, les empreintes numériques de *Payload* et *info* sont des chaînes hexadécimales de 64 caractères (256 bits) générées par le hash SHA-256 démontré dans la partie f de la partie 1, car la fonction *split_ secret* de la bibliothèque secretsharing attend un secret sous forme de chaîne hexadécimale, suite à cela, lors de cette première étape, la bibliothèque utilisée (secretsharing) convertit en interne les chaînes hexadécimales en nombre(s) dans un corps fini pour obtenir un vecteur numérique pour *Payload* et *info* afin d'appliquer Shamir.

Donc, soit un secret vectoriel

$$\mathbf{s} = (s_1, s_2, \ldots, s_L) \tag{1}$$

de longueur L (64), avec chaque s_j appartenant à un corps fini $\mathbb{F}_q$.

On souhaite distribuer ce secret à n participants selon un seuil t, en construisant pour chaque élément s_j un polynôme aléatoire de degré $t - 1$ dont le terme constant est s_j :

$$f_j(x) = s_j + a_{j,1}x + a_{j,2}x^2 + \cdots + a_{j,t-1}x^{t-1} \tag{2}$$

où $a_{j,k} \in \mathbb{F}_q$ sont choisis aléatoirement.

On choisit n points distincts non nuls $x_1, x_2, \ldots, x_n \in \mathbb{F}_q \setminus \{0\}$. Pour chaque participant $i = 1, \ldots, n$, la part distribuée est le vecteur :

$$M_i = (f_1(x_i), f_2(x_i), \ldots, f_L(x_i)) \tag{3}$$

Autrement dit, la matrice des parts M est de dimension $n \times L$ et s'écrit :

$$M = \begin{pmatrix} f_1(x_1) & f_2(x_1) & \cdots & f_L(x_1) \\ f_1(x_2) & f_2(x_2) & \cdots & f_L(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ f_1(x_n) & f_2(x_n) & \cdots & f_L(x_n) \end{pmatrix} \tag{4}$$

Une fois les parts M obtenues de *Payload* et *info*, elles sont écrites dans les fichiers respectifs, *shamirPartsPayload.txt* et *shamirPartsInfo.txt*.

8.3 Deuxième étape : César généralisé appliqué à une matrice numérique

Dans cette deuxième partie du chiffrement, le chiffrement de César est appliqué à chaque valeur de chaque part, la différence avec la vraie fonction de César est qu'à la place de manipuler des lettres dans un alphabet cyclique, le décalage est réalisé sur chaque cellule des matrices *Payload* et *info* sur des valeurs numériques.

Premièrement, une valeur entre 100 et 999 est attribuée à la variable *caesarValue* de la manière suivante :

```
caesarValue = (rand() %999 ) + 100;
```

La fonction *caesar* est appelée avec comme argument *shamirBlendedCaesarPayload/info* qui est le résultat de la fonction *shamirpartsPayload/info* qui contient les parts et *caesarValue* la valeur de César.

```
caesar(shamirBlendedCaesarPayload,shamirpartsPayload,caesarValue);
caesar(shamirBlendedCaesarInfo,shamirpartsInfo,caesarValue);
```

```
void caesar(int tab[ROWS][COLS], int tabShamir[ROWS][COLS],int caesarValue) {
  int limit = 256;
  for (int i = 0; i < ROWS; i++) {
    for (int j = 0; j < COLS; j++) {
        int index = tabShamir[i][j] + caesarValue;
        int validIndex = index % limit;
        tab[i][j] = validIndex;
    }
  }
}
```

Figure 21 – *Script de la fonction caesar*

La fonction *caesar* sur la figure 21 permet donc cette deuxième étape de chiffrement, voici son fonctionnement : une variable *(limit)* a pour valeur 256 afin que les valeurs soient comprises dans un espace $[0, 255]$, puis, chaque valeur numérique de la matrice M de l'équation 4 se voit appliquer la valeur de *caesarValue*, enfin le modulo appliqué à la variable *index* permet ainsi que les nouvelles valeurs pour chaque s_i de l'équation 1 soient calculables sur 8 bits.

La seconde partie de cette deuxième étape consiste en la conversion binaire des valeurs décimales par la fonction *decToBinary*.

```
decToBinary(ceasarBinaryPayload,shamirBlendedCaesarPayload);
decToBinary(ceasarBinaryInfo,shamirBlendedCaesarInfo);
```

ainsi chaque vecteur numérique a maintenant une longueur de 256 octets, soit 512 bits, les variables *caesarBinaryPayload* et *caesarBinaryInfo* sont déclarées de la manière suivante :

```
int ceasarBinaryPayload[ROWS][COLS][BITS];
int ceasarBinaryInfo[ROWS][COLS][BITS];
```

avec les variables constantes :

```
#define ROWS 8
#define COLS 64
#define BITS 8
```

Afin d'avoir un aperçu visuel du traitement des données depuis la génération d'une empreinte numérique jusqu'à sa forme binaire, un exemple est traité sur la figure 22

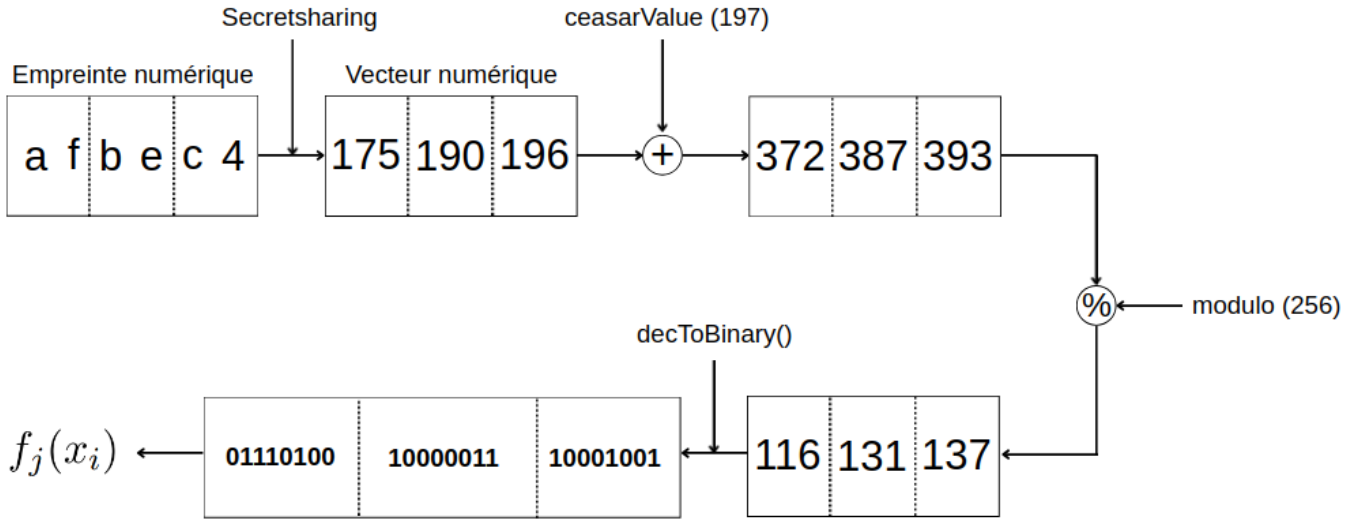

Figure 22 – *Traitement des données jusqu'à la deuxième étape*

Pour ainsi expliquer le déroulement du traitement des données depuis la première étape, l'empreinte numérique est générée, le traitement du chiffrement commence alors par l'entrée de cette empreinte numérique de 64 caractères héxadécimaux dans *secretsharing* qui transforme cette chaîne en vecteur numérique afin de pouvoir être traité. La valeur de caesarValue est initialement assignée, suivie de l'application du modulo pour obtenir $f_j(x_i)$. Enfin, la fonction binaire renvoie chaque part de 512 bits pour obtenir la matrice M de l'équation 4, donc au final, sachant que 8 parts sont générées, alors la matrice doit contenir $(L \cdot b) \cdot n$ bits, ou b représente la valeur décimale du nombre de bits nécessaires sur laquelle la variable *limit* peut avoir une valeur binaire, dans ce cas-ci, la matrice M contient donc 4096 bits.

8.4 Troisième étape : Application d'un masque binaire

Cette troisième et dernière étape du chiffrement consiste à appliquer un masque binaire sur chaque bit de la matrice M, pour ce faire, il est attribué une valeur aléatoire comprise entre $[1, 255]$ à la variable *maskValue*, puis cette valeur est convertie de sa forme décimal à sa forme binaire à 8 bits dans la variable *maskBits*. Après cela, la fonction *xor* visualisable sur la figure 23 est exécutée afin d'appliquer le masque.

Pour procéder ainsi, pour chaque 8 bits, c'est par rapport à la position k que le bit du masque est choisi, un exemple est démontré sur la figure 24.

```
void xor(int tab[ROWS][COLS][BITS], int tabBinary[ROWS][COLS][BITS], unsigned char *maskBits) {

  for (int i = 0; i < ROWS; i++) {
    for (int j = 0; j < COLS; j++) {
      for (int k = 0; k < BITS; k++) {
        tab[i][j][k] = tabBinary[i][j][k] ^ maskBits[k];
      }
    }
  }
}
```

Figure 23 – *Script de la fonction xor*

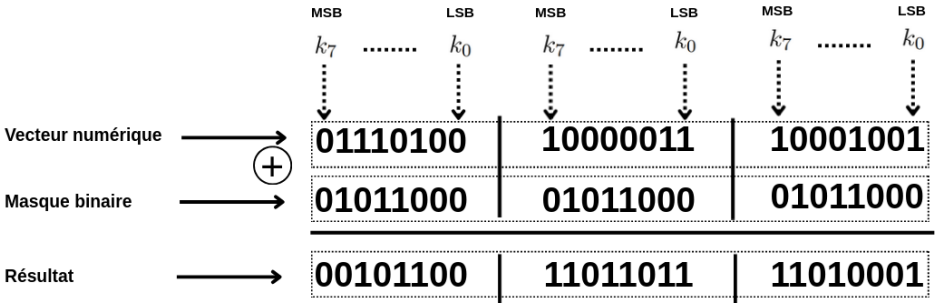

Figure 24 – *Application du masque binaire*

Afin de savoir comment interpréter le XOR utilisé par le masque binaire sur le vecteur binaire obtenue par la fonction *binary* visible sur la figure 22, voici l'explication :

Sur chaque séquence de 8 bits en commençant par la droite, est appliqué le masque binaire de la manière suivante. Lorsque $k == 0$ alors c'est le premier bit du masque binaire *maskBits[0]*, soit *0* qui est appliqué sur le LSB du vecteur numérique (Least Significant Bit) qui a la valeur *1*. Avec le tableau 3 il est possible de déterminer que le résultat obtenu est *1*, ceci est à faire jusqu'à ce que $k == 7$, ce qui termine cette séquence de 8 bits sur le MSB (Most Significant Bit) et ainsi de suite jusqu'à la fin de chaque vecteur binaire de la matrice M. Cette conversion XOR peut être décrite avec l'équation suivante pour sa réalisation sur toute la matrice :

$$\forall i, j, \quad \text{XOR}(f_j(x_i)) = f_j(x_i) \oplus B_{i,j} \tag{5}$$

ou B représente le masque binaire.

Table 3 – *Tableau XOR*

X	Y	$S = X \oplus Y$
1	*1*	*0*
1	*0*	*1*
0	*1*	*1*
0	*0*	*0*

La figure 25 reprend la sortie $f_j(x_i)$ du schéma *Traitement des données deuxième étape* représenté sur la figure 22. Il est donc possible de voir la fonction *xor* appliquée sur la matrice M. Enfin, la fonction *send* se charge d'écrire les deux matrices générées dans le fichier *CryptFile.txt*, (elles seront appelées $M_{Payload}$ et M_{Info} pour la suite de ce rapport) ainsi que la valeur de césar et la valeur décimale du masque binaire, le contenu du fichier *CryptFile.txt* doit donc ressembler au fichier de la figure 26, le fichier peut contenir entre 8208 et 8210 bits (calculé avec la formule 6) dans le cas des valeurs utilisées dans ce rapport, ce qui représente 1026 octets soit seulement 1 kilooctet, ce qui est extrêmement léger et permet donc un transfert rapide des données et également de traitement.

$$\Big(\big((L \cdot b) \cdot n\big) \cdot 2\Big) + \text{caesarValue} + \text{maskValue} \tag{6}$$

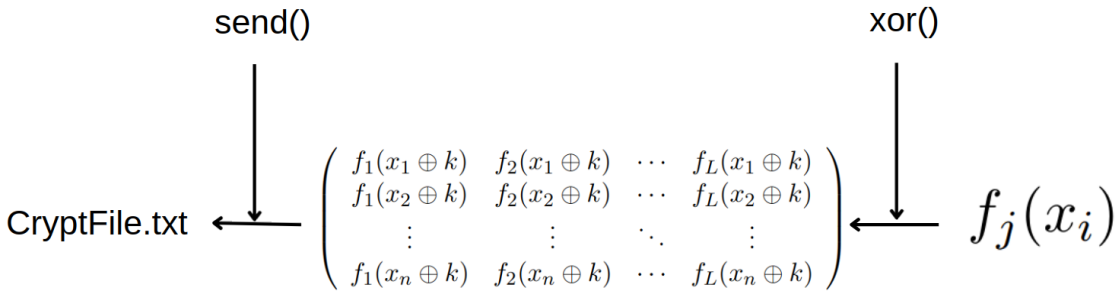

Figure 25 – *Traitement XOR jusqu'à écriture dans le fichier CryptFile.txt*

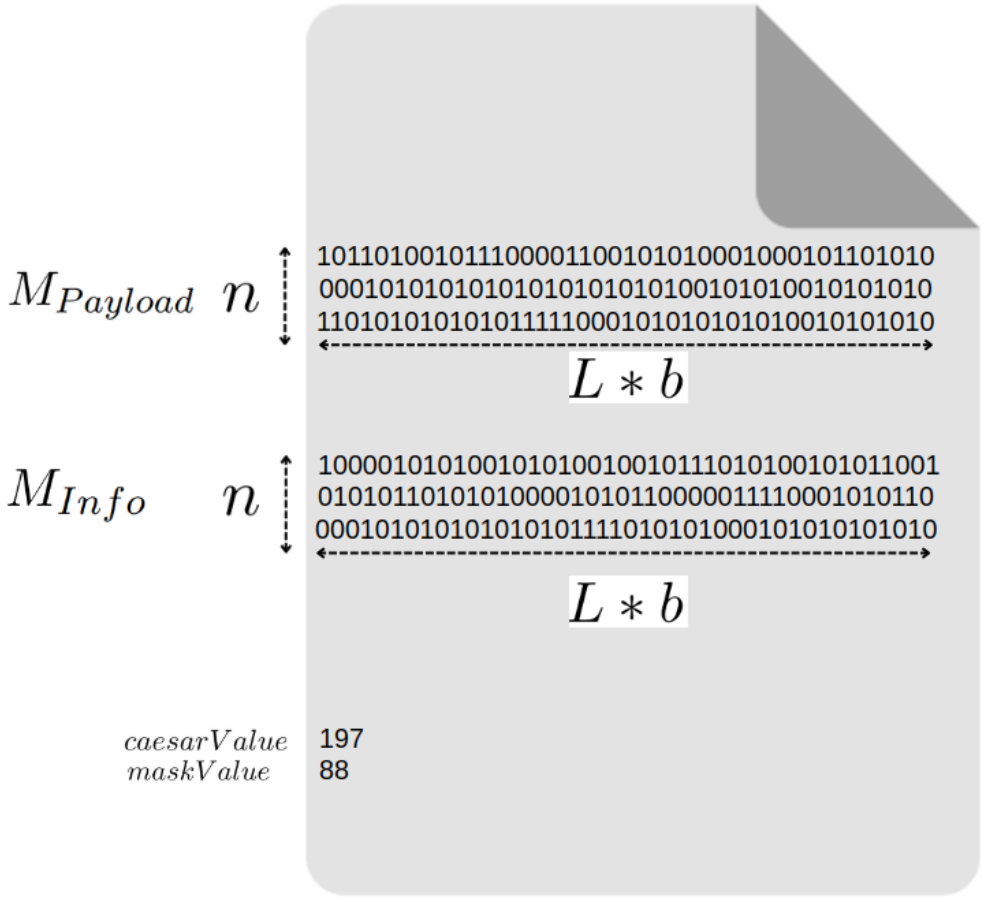

Figure 26 – *Fichier CryptFile.txt*

9 Partie 3 : Transfert des données

Cette partie 3 se concentre sur le transfert des données, le protocole utilisé, l'explication de son choix ainsi que ses avantages. Une vue d'ensemble est représentée sur la figure 27.

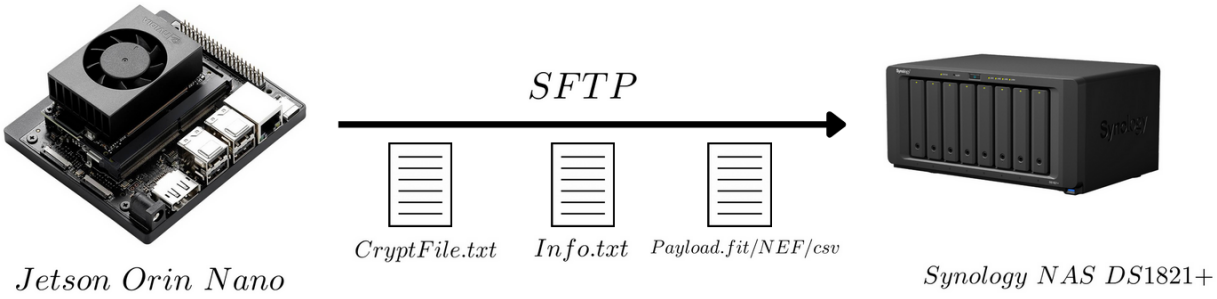

Figure 27 – *Vue d'ensemble du transfert des données*

9.1 Raisons ayant conduit à l'adoption du protocole SFTP

Plusieurs facteurs devaient être pris en considération pour l'adoption du protocole :

- Sécurité
- Portabilité
- Intégrité
- Rapidité

En effet, SFTP utilise le chiffrement fourni par SSH pour sécuriser les données en transit, ce qui lui permet de sécuriser à la fois les données, les commandes et les identifiants pendant le transfert. Les différentes commandes utilisées pour le transfert sont : *cd* et *put* pour respectivement se déplacer dans le répertoire de dépôt des fichiers du NAS, et l'envoi des fichiers.

Grâce à sa portabilité, SFTP peut être utilisé sur n'importe quelle plateforme qui supporte SSH, y compris Windows, Linux, et macOS.

SFTP utilise des algorithmes de hachage tel que SHA-2 pour vérifier que les fichiers n'ont pas été modifiés pendant le transfert, garantissant ainsi leur intégrité.

```
if [[ "$count" != 5 ]]; then
  lftp -u $USER,$PASSWD sftp://$HOST <<EOF
  set sftp:auto-confirm yes
  echo "int the case count is not equal 5"
  cd "$REMOTE_DIR"
  put -- "$NAMEfilePayload"
  put -- "$FileInfo"
  put -- "$CryptFile"
  bye
  exit
EOF
fi
```

Figure 28 – *Script de l'envoie des fichiers*

La figure 28 montre la manière dont les fichiers sont envoyés. La CLI (Command Line Interface) *lftp* permet ainsi de se connecter au NAS et d'envoyer les fichiers dans l'ordre suivant : *Payload*, *Info* et *CryptFile.txt*.

Également, un autre moyen pour transférer les fichiers pourrait être avec *curl*, mais avec les tests ci-dessous, il est démontré pourquoi *lftp* a été favorisé en comparant les protocoles SFTP et FTPS (File Transfer Protocol Secure).

9.1.1 Benchmarking Download et Upload SFTP/FTPS

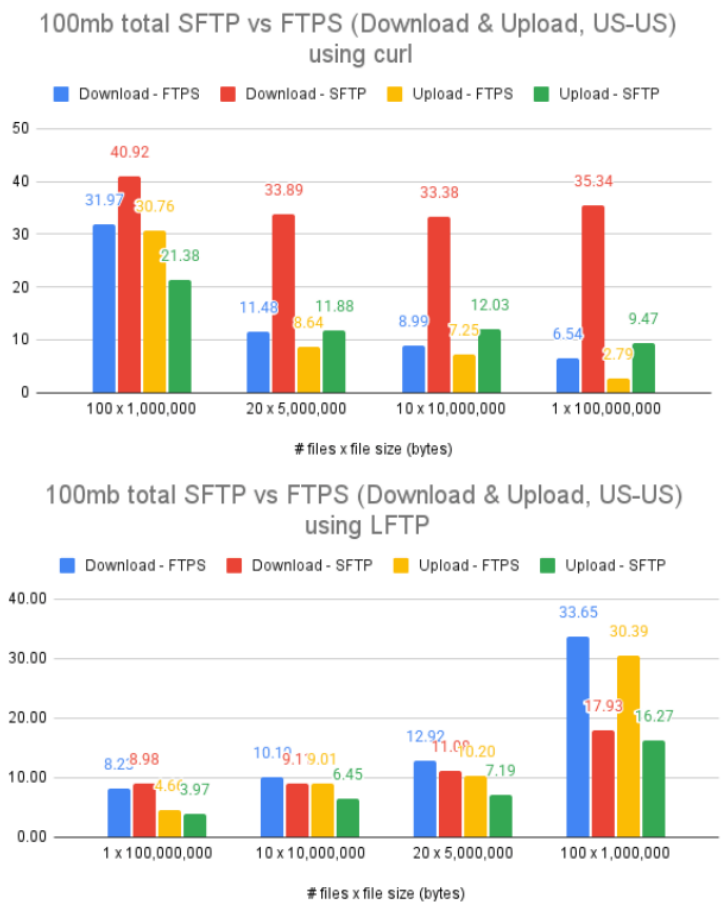

Figure 29 – *Comparaison vitesse de transferts des CLI lftp et curl, capture d'écran réalisée à partir du site [sftptogo.com], consulté le [13 Mars 2025].*

Afin d'avoir quelques points de comparaison entre les Téléchargements (Download) et les Téléversements (Upload) des deux protocoles différents, SFTP et FTPS utilisant chacun d'eux les CLI lftp et curl. Premièrement, il est à prendre en compte que les commandes qui constituent le batch [8] (traitement par lots) représentent environ 54, 5 MB de données à transférer. Sur la figure 29, sont représentés sur la partie du haut quatre différents tests réalisés avec la CLI *curl* pour envoyer un total de 100 MB. En les comparant à la partie du bas où les données sont traitées avec la CLI $lftp$, il est largement constatataté que dans l'ensemble des cas la CLI $lftp$ permet de meilleures performances, que ce soit dans le cas de downlink que dans le cas d'uplink. Il est donc clair sachant que les commandes nécessaires sont manipulables par la CLI $lftp$ que le choix soit orienté vers celui-ci.

En ce qui concerne la sélection du protocole à sélectionner, les tests du graphique du bas sont donc plus cohérents et sont ainsi pris en considération. Dans les quatre différents cas, le protocole $SFTP$ performe mieux. Il est important de noter que plus de fichiers sont envoyés, plus l'écart entre les deux protocoles est important alors que la quantité de données envoyées est égale, ce qui suggère que le nombre de batch impacte plus le protocole $FTPS$ (sûrement dû au certificat SSL/TLS (visualisable sur le tableau des comparaisons) 4)) que le protocole $SFTP$, sachant que l'envoi des données de la station se réalise à chaque fois que des données sont reçues, et que chaque équipement capture des données à un intervalle de 30 secondes qui correspond à un rythme soutenu, alors le nombre de batch a un impact important à prendre en considération.

Pour appuyer davantage cette tendance, la figure 30 représente l'envoi de différentes tailles de données pour le même nombre de fichiers en utilisant la CLI $lftp$. Il est constaté que l'uplink, qui est la valeur qui nous intéresse, a le temps le plus faible pour réaliser les envois.

Également, il y a le protocole FTP (File Transfer Protocol), qui permet une vitesse de transfert plus rapide, mais celui-ci ne fournit pas de chiffrement. Cela signifie que toute personne interceptant le trafic peut lire les données et potentiellement accéder au serveur, contrairement au protocole SFTP qui permet de remédier au mieux à cela.

8. Désigne l'exécution automatique d'une série de commandes ou de tâches, sans intervention humaine pendant le processus.

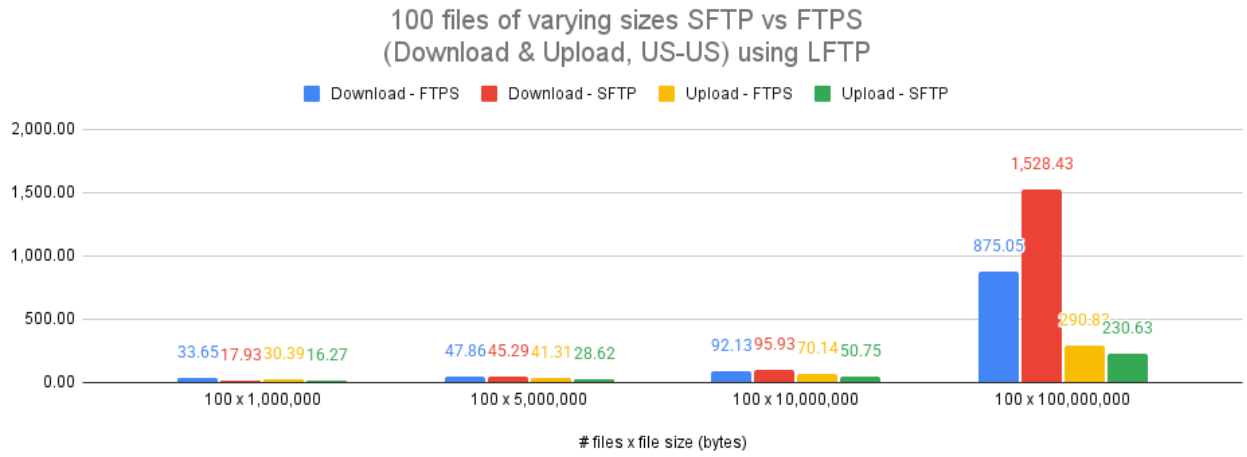

Figure 30 – *Envoie de différentes tailles, capture d'écran réalisée à partir du site [sftptogo.com], consulté le [13 Mars 2025].*

Table 4 – *Comparaison entre les protocoles FTP, FTPS et SFTP*

	FTP	FTPS	SFTP
1. Force de la norme de cryptage des données	✗	✓	✓
2. Chiffrement des login/mdp	✗	✓	✓
3. Authentification par clé	✗	✗	✓
4. Certificat SSL/TLS	✗	✓	✗
5. Compatibilité avec les pare-feu	✗	✗	✓

Avec ceci dit, le protocole *SFTP* est actuellement celui utilisé afin d'assurer la meilleure rapidité tout en assurant une intégrité et confidentialité des données. Cependant, avec le système d'élimination de fichiers développé dans la partie 7.6 si la confidentialité n'était pas prise en compte, alors le protocole *FTP* serait celui à privilégier.

9.2 Tailscale

Afin que la station puisse être capable d'envoyer les données vers le NAS, la création d'un réseau privé virtuel (tailnet) a été réalisée avec Tailscale. Tailscale permet l'envoi de données entre un PC (Personal Computer) et un NAS sur des réseaux différents grâce à sa technologie de VPN (Virtual Private Network) maillé basée sur WireGuard. Ce qui permet de connecter tous les appareils à un réseau privé sécurisé, peu importe leur emplacement physique ou leur réseau local. Chaque appareil reçoit une adresse IP stable dans ce réseau virtuel (voir figure 31), ce qui permet une communication directe. De plus, étant donné que les données transitent via des tunnels chiffrés WireGuard, cela assure la sécurité et la confidentialité. Concernant la connexion directe peer-to-peer entre les appareils, C'est un point très intéressant à prendre en compte pour le futur du projet quand les stations auront besoin de communiquer directement entre elles.

À part cela, d'autres fonctionnalités peuvent être prises en considération pour le futur du projet :

- **MagicDNS** : Permet d'utiliser des noms de domaine lisibles (par exemple `station1.tailnet`) au lieu d'adresses IP, ce qui simplifie la communication entre appareils.
- **ACL (Access Control Lists)** : Offre la possibilité de définir précisément quels appareils ou utilisateurs ont accès à quels services, renforçant ainsi la sécurité du réseau.
- **Subnet routers** : Si le NAS se trouve sur un réseau local non compatible avec Tailscale (ce qui n'est pas le cas pour le moment), il est possible d'utiliser un routeur Tailscale pour exposer ce réseau au reste du *tailnet*.

— **Exit nodes** : Permet de faire transiter l'intégralité du trafic Internet d'un appareil via un autre, ce qui peut être utile pour centraliser les connexions ou contourner certaines restrictions réseau.

— **Tailscale SSH** : Fournit une méthode sécurisée pour se connecter en SSH à distance, sans nécessiter la gestion manuelle des clés.

— **Audit et journalisation** : Propose des journaux de flux réseau et de configuration, utiles pour assurer le suivi et la traçabilité des échanges entre stations.

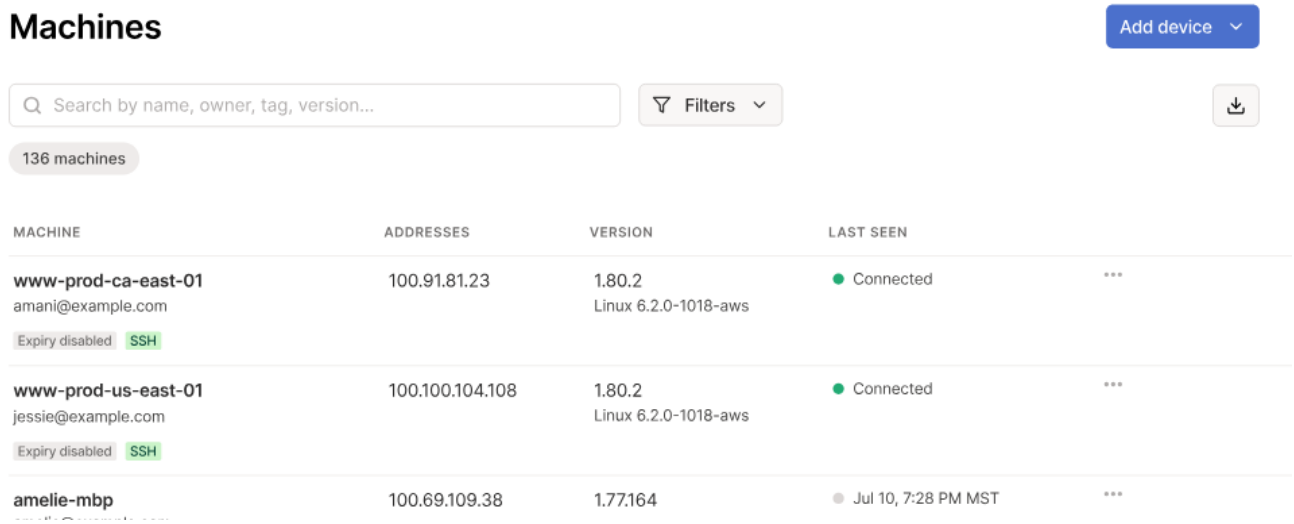

Figure 31 – *Interface de Tailscale, capture d'écran réalisée à partir du site [tailscale.com], consulté le [17 Mars 2025].*

10 Partie 4 : Traitement des données lors de leur réception

Dans cette quatrième partie, les données reçues par le NAS sont traitées, et leur organisation est appliquée une fois leur intégrité vérifiée. Les parties a à d du diagramme 32 sont donc expliquées indépendamment dans cette quatrième partie.

10.1 Diagramme général et explication

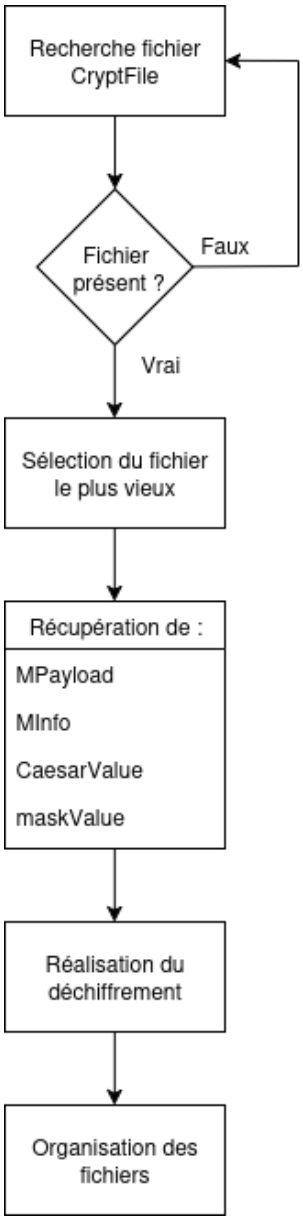

Figure 32 – *Diagramme Partie 4*

La partie a représente la recherche d'au moins un fichier *CryptFile.txt* par le NAS dans chaque répertoire d'équipement utilisé pour chaque station, si un fichier est détecté dans le répertoire qui peut être par exemple : */var/services/homes/INDICATIC/Pacifico/INDICATIC/UTP/ALPY* pour le cas d'Alpy, alors le fichier le plus ancien est sélectionné. À partir de là, il est écrit dans un fichier indépendant pour chaque variable, les valeurs de $M_{Payload}$, M_{Info}, $caesarValue$ ET $maskValue$ (partie b). La partie c, quant à elle, traite le déchiffrement à partir de ces quatre fichiers qui contiennent les variables requises pour procéder et obtenir l'empreinte numérique (appelée secret dans la partie chiffrement (8)) de *Payload* et *Info*. Enfin, la partie d va gérer l'organisation des fichiers dans le cas où les empreintes numériques obtenues sont les mêmes ou pas générées par le NAS.

10.2 Partie a : Recherche de fichier CryptFile existant

Dans cette partie a le NAS recherche donc les fichiers *CryptFile.txt*, afin d'avoir une meilleure compréhension de cette première étape de la partie 4, voici l'explication du diagramme de la figure 33 :

Premièrement, le script *main.sh* permet l'exécution du code d'interopérabilité côté NAS. La première fonctionnalité que fait ce script est d'exécuter le script *OrderImages.sh* qui permet l'organisation des fichiers une fois leur intégrité vérifiée. Afin de déterminer si un fichier $CryptFile.txt$ est présent, le code va commencer à rechercher dans chaque répertoire de la station *caribe* puis, une fois chaque répertoire d'une station vérifié, il est procédé à la même méthode dans autant de répertoires différents que le NAS contient. Quand un fichier $CryptFile.txt$ est détecté, le fichier $fileTreament.sh$ est exécuté avec comme argument le répertoire où se trouve le fichier, la ligne de code ci-bas permet ceci :

```
directory="/var/services/homes/INDICATIC/"$dir"/ALPY"
/var/services/homes/INDICATIC/InteroperabilityCode/fileTreatment.sh "$directory"
```

Une fois le script *fileTreament.sh* exécuté avec l'argument, le premier traitement réalisé est de détecter le fichier *CryptFile.txt* le plus ancien et donc de repérer les fichiers *Payload* et *Info* associés.
Le code ci-dessous permet ainsi cela.

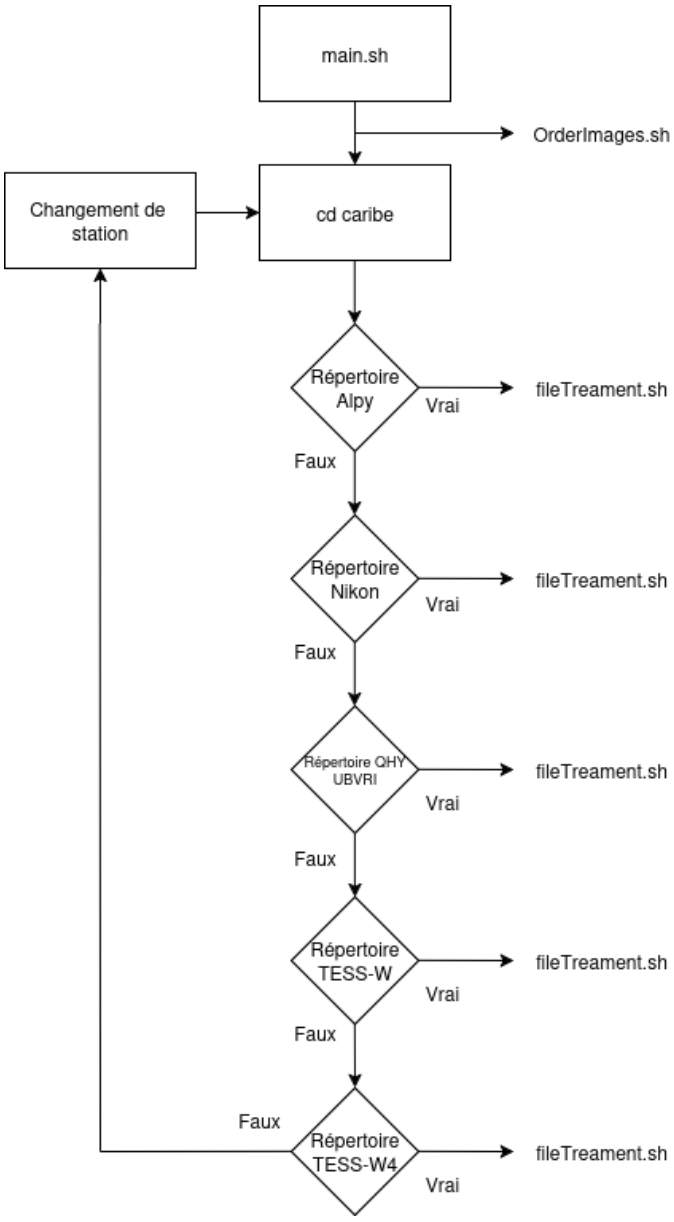

Figure 33 – *Diagramme Partie a*

```
FindFiles=$(find . \( -type f \) -name
↪  "*CryptFile*")
name=$(ls -lt $FindFiles | tail -n 1 |
↪  awk '{print $NF}')$
```

Le nom du fichier sélectioné est ainsi associé à la variable, *FindFiles* et la variable *name* sélectionne le fichier le plus ancien. Par la suite les deux lignes de code suivante :

```
FileName=$(echo "$name" | awk
↪  '{sub(".txt$", "", $0); print $0}')
FileName=$(echo "$FileName" | awk -F
↪  '_' '{print substr($0, index($0,
↪  "_") + 1)}')
```

permettent de seulement contenir le nom du fichier dans la variable *FileName*, par exemple si le fichier a pour nom *CryptFile_wed123.txt* alors la variable *FileName* contiendra seulement *wed123*, c'est avec ce nom que le NAS peut associer les fichiers *Payload* et *info* correspondants au bon fichier *CryptFile.txt*.

10.3 Partie b : Récupération des valeurs du fichier CryptFile.txt

Cette partie traite donc une étape cruciale afin que le code de déchiffrement puisse avoir accès aux données du fichier *CryptFile.txt*, comme il est possible de le voir dans les lignes de code ci-dessous. La variable *name* qui contient le nom du fichier *CryptFile* mentionné plus haut dans la sous-section 10.2 est utilisée afin d'écrire $M_{Payload}$, M_{Info} $caesarValue$ et $maskValue$ dans un fichier respectif pour chacun.

```
cat "$name" | tr '\n' ' ' | awk '{print $1}' >
↪ /var/services/homes/INDICATIC/InteroperabilityCode/tmpFile/BitPayloadToSend.txt
cat "$name" | tr '\n' ' ' | awk '{print $2}' >
↪ /var/services/homes/INDICATIC/InteroperabilityCode/tmpFile/BitInfoToSend.txt
cat "$name" | tr '\n' ' ' | awk '{print $3}' >
↪ /var/services/homes/INDICATIC/InteroperabilityCode/tmpFile/Caesar.txt
cat "$name" | tr '\n' ' ' | awk '{print $4}' >
↪ /var/services/homes/INDICATIC/InteroperabilityCode/tmpFile/Mask.txt
```

10.4 Partie c : Réalisation du déchiffrement

La ligne de code :

```
/var/services/homes/INDICATIC/InteroperabilityCode/mainTreatment
```

permet d'exécuter le code principal de déchiffrement, une fois que le déchiffrement est terminé, les empreintes numériques de *Payload* et *Info* sont écrites dans le fichier *HashFile.txt* puis sont récupérées dans la suite du traitement des données :

```
HashFile="/var/services/homes/INDICATIC/InteroperabilityCode/Hash.txt"
```

Enfin, les valeurs des empreintes numériques sont attribuées aux variables *PayloadHash* et *InfoHash*, ces variables seront comparées avec les empreintes numériques générées par le NAS sur les fichiers *Payload* et *Info* exactement de la même manière que la station, soit avec *sha256*.

```
read -r PayloadHash < "$HashFile"
read -r InfoHash < <(tail -n +2 "$HashFile"
```

10.5 Partie d : Organisations des fichiers

Cette dernière partie du traitement des fichiers concerne l'organisation une fois les empreintes numériques générées avec *sha256* et obtenues avec le déchiffrement sont comparées voir figure 34. Dans le cas où les empreintes numériques sont les mêmes, alors seulement le fichier *CryptFile.txt* est supprimé et les fichiers *Payload* et *Info* sont transférés dans un dossier créé et nommé par le même nom que la variable *name*. Dans le cas où les empreintes ne sont pas identiques, alors les 3 fichiers transférés sont supprimés.

Concernant le script *OrderImages.sh*, celui-ci vérifie périodiquement la présence des dossiers créés dans chaque répertoire. S'il en trouve un, alors étant donné que ce dossier contient les deux fichiers avec l'intégrité vérifiée, alors ils sont transférés dans le dossier où tous les fichiers vérifiés se trouvent et le dossier créé spécialement pour les deux fichiers est supprimé.

Figure 34 – *Vérifications des empreintes numériques*

11 Partie 5 : Déchiffrement des empreintes numériques

Dans cette partie 5 sont expliquées les trois différentes étapes de déchiffrement réalisées par le NAS à partir des informations du fichier *CryptFile.txt* reçues afin de déterminer les empreintes numériques des fichiers *Payload* et *Info*. Il est d'abord expliqué le diagramme général de la partie, puis les trois différentes étapes réalisées.

11.1 Diagramme général et explication

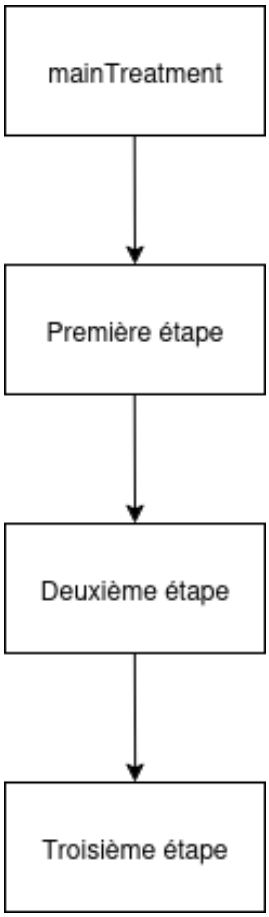

Figure 35 – *Diagramme Partie 5*

Sur la figure 35 sont représentées les trois étapes de déchiffrement nécessaires pour retrouver les empreintes numériques de *Payload* et *Info*. La première étape consiste à appliquer l'opération inverse du XOR avec la valeur de la variable $maskValue$ du fichier reçu *CryptFile.txt*, ainsi que de réaliser la conversion du résultat obtenu de binaire à décimal. La seconde étape consiste à réaliser l'inverse du chiffrement de César avec la valeur de $caesarValue$ obtenue. La dernière étape consiste en l'utilisation de l'interpolation lagrangienne afin de retrouver la valeur des empreintes numériques.

11.2 Première étape : Opération inverse du XOR

Cette première étape consiste simplement à réaliser l'opération inverse du XOR sur la matrice M (M peut représenter soit $M_{Payload}$ soit M_{Info}), la réalisation inverse du XOR consiste à réaliser de nouveau exactement la même démarche que la troisième étape (8.4) de la partie chiffrement, en reprenant le résultat obtenu et en appliquant le masque binaire, le vecteur numérique obtenu est 011101001000001110001001 puis est ensuite converti en décimal.

11.3 Deuxième étape : Opération inverse de César

Pour rappel, dans la partie chiffrement, l'opération de César est réalisée de la manière suivante :

```
int index = tabShamir[i][j] + caesarValue;
int validIndex = index % limit;
tab[i][j] = validIndex;
```

Donc, afin de retrouver la valeur décimale avant l'application de César, l'opération suivante est utilisée :

```
tabShamir[i][j] = ((tab[i][j] - caesarValue + limit) % limit);
```

En repartant du résultat obtenu après l'opération inverse du XOR qui est 116 | 131 | 137, le résultat obtenu est 175 | 190 | 196.

11.4 Troisième étape : Interpolation lagrangienne

Dans cette troisième étape, pour reconstruire le secret, il est utilisé la **formule d'interpolation de Lagrange**, qui permet de retrouver la valeur du polynôme en zéro (i.e. le terme constant, donc le secret) à partir de t points (x_i, y_i) :

$$f(0) = \sum_{i=1}^{t} y_i \cdot \ell_i(0) \tag{7}$$

Où chaque coefficient de Lagrange $\ell_i(0)$ est défini par :

$$\ell_i(0) = \prod_{\substack{j=1 \\ j \neq i}}^{t} \frac{0 - x_j}{x_i - x_j} \tag{8}$$

Cette expression calcule le poids de chaque fragment dans la reconstruction du polynôme à zéro, en excluant les autres indices pour chaque i.

Afin de démontrer un exemple, il est utilisé un partage de secret de Shamir avec $n = 8$ fragments et un seuil $t = \frac{n}{4} = 2$. Le secret à partager est $S = 123$, qui est encodé dans le terme constant du polynôme :

$$f(x) = a_0 + a_1 x = 123 + 45x \tag{9}$$

Sont générés les fragments en évaluant ce polynôme pour $x = 1$ à $x = 8$, ce qui donne les points suivants :

$$\begin{aligned} &(1, 168), \quad (2, 213), \quad (3, 258), \quad (4, 303), \\ &(5, 348), \quad (6, 393), \quad (7, 438), \quad (8, 483) \end{aligned} \tag{10}$$

Pour reconstruire le secret, il est choisi deux points au hasard parmi ceux disponibles, par exemple :

$$(x_1, y_1) = (2, 213), \quad (x_2, y_2) = (5, 348) \tag{11}$$

Il est utilisé l'interpolation de Lagrange pour calculer $f(0)$, ce qui correspond au secret :

$$f(0) = y_1 \cdot \frac{0 - x_2}{x_1 - x_2} + y_2 \cdot \frac{0 - x_1}{x_2 - x_1} \tag{12}$$

Calculs :

$$\begin{aligned} f(0) &= 213 \cdot \frac{-5}{2 - 5} + 348 \cdot \frac{-2}{5 - 2} \\ &= 213 \cdot \frac{-5}{-3} + 348 \cdot \frac{-2}{3} \\ &= 213 \cdot \frac{5}{3} + 348 \cdot \left(-\frac{2}{3}\right) \\ &= \frac{1065}{3} - \frac{696}{3} \\ &= \frac{369}{3} \\ &= 123 \end{aligned} \tag{13}$$

Donc, le secret reconstitué est :

$$\boxed{123}$$

Afin d'avoir un aperçu visuel, la procédure complète du déchiffrement est représentée sur la figure 36.

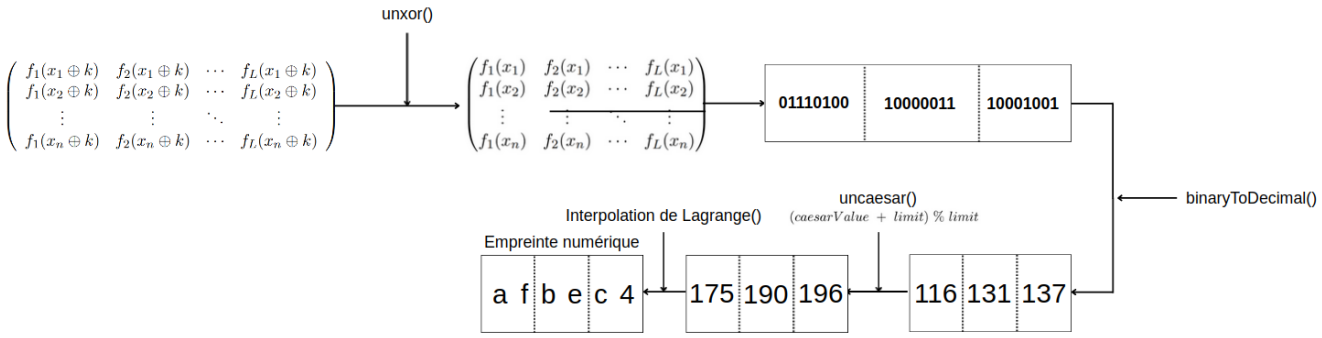

Figure 36 – *Traitement des données déchifrement*

Le résultat obtenu est ainsi l'empreinte numérique de *Payload* et *Info* qui sont ensuite comparées avec les empreintes numériques générées par le NAS (expliqué dans la partie 10.5).

12 Partie 6 : Interface, lancement et contrôle de la station

Cette dernière portion du code d'interopérabilité prend en charge l'automatisation intégrale de la station, en assurant son fonctionnement de façon autonome. Différents paramètres, détaillés ci-dessous, sont contrôlés afin de garantir la connectivité des appareils reliés à la station durant l'exécution des cycles automatisés. Une logique spécifique a également été mise en place pour permettre la répétition programmée de ces cycles à des horaires prédéfinis. Par la suite est démontré le contrôle de la station en cas de différents scénarios possibles tels que, coupure d'internet, mémoire de la station proche de la saturation.

12.1 Interface et lancement

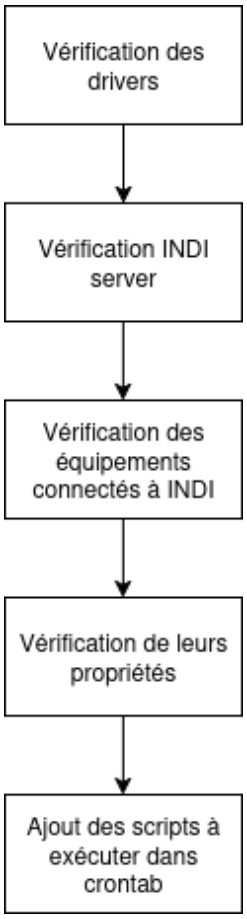

Figure 37 – *Diagramme Partie 6*

— **Vérification des drivers :** La première vérification effectuée est la vérification des drivers, avec la commande :

```
lsusb > /tmp/drivers.txt
drivs=$(cat /tmp/drivers.txt)
```

il est possible de savoir les équipements qui sont bien connectés à la station avec le contenu de la variable *drivs*, pour chaque équipements manquant un email d'avertissement est envoyé.

— **Vérification INDI server :** La vérification de INDI server permet de détecter si le serveur INDI est déjà exécuté ou non, dans le cas échéant, alors celui-ci est lancé et la présence de détection des équipements est vérifiée (voir figure 6) et confirmée, dans le cas contraire un email d'avertissement est envoyé.

— **Vérification des équipements connectés à INDI :** Suite aux vérifications du serveur, il est vérifié la présence des propriétés des équipements (voir figure 7), si aucune propriété pour un équipement n'est détectée, alors un email d'avertissement est envoyé.

— **Ajout et vérification Crontab** [9] **:** Afin d'automatiser la station, toutes les différentes tâches sont planifiées dans Crontab où Cron est le démon (service) qui va se charger d'exécuter les planifications. Ainsi, cette automatisation des tâches est organisée en deux parties : la partie de l'exécution des tâches, et la partie de l'arrêt des tâches comme il est démontré sur la figure 38. Sur cette figure, il est visualisable que chaque ligne commence par des variables de couleur blanche, ces variables ont pour valeurs le début et la fin d'un cycle ou *Begin* signifie un début de cycle et *Kill* signifie la fin de cycle. Les lignes de code ci-dessous montrent comment sont attribuées les valeurs pour définir le temps du cycle :

```
#Default time to begin a cycle
BegingDefaultAllHour="14"
BegingDefaultAllMinute="35"

#Default time to terminate a cycle
EndDefaultAllHour="16"
EndDefaultAllMinute="0"
```

9. Crontab est un outil qui permet de lancer des tâches de façon régulière sur les systèmes Linux.

Puis, par la suite ces valeurs sont attribuées de la manière suivante :

```
BegingCodeAlpyHour="$BegingDefaultAllHour"
BegingCodeAlpyMinute="$BegingDefaultAllMinute"
KillCodeAlpyHour="$EndDefaultAllHour"
KillCodeAlpyMinute="$EndDefaultAllMinute"
```

```
$BegingCodeQHYMinute $BegingCodeQHYHour * * * /home/indicatic-e1/Desktop/INDIcode/qhy > /tmp/logQHY.txt 2>&1 & echo \$! > /tmp/qhy_ccd_test.pid
$BegingCodeAlpyMinute $BegingCodeAlpyHour * * * /home/indicatic-e1/Desktop/INDIcode/alpy > /tmp/logalpy.txt 2>&1 & echo \$! > /tmp/my_client.pid
$BegingCodeNikonMinute $BegingCodeNikonHour * * * /home/indicatic-e1/Desktop/code/nikon.sh > /tmp/lognik.txt 2>&1 & echo \$! > /tmp/nikon.pid
$BegingCodeInteropMinute $BegingCodeInteropHour * * * [ ! -s /tmp/Interop.pid ] || ! grep -q '[0-9]' /tmp/Interop.pid && /home/indicatic-e1/Desktop/AutoRun/GetInteropCodePID.sh
$BegingConverterPYMinute $BegingConverterPYHour * * * python3 /home/indicatic-e1/Desktop/code/converter.py > /tmp/logConverter.txt 2>&1 & echo \$! > /tmp/Converter.pid
$BegingCodeTESSMinute $BegingCodeTESSHour * * * /home/indicatic-e1/Desktop/code/Interop_code/TESS.sh > /tmp/logTESS.txt 2>&1 & echo \$! > /tmp/TESS.pid
$BegingCheckAlpyLogsMinute $BegingCheckAlpyLogsHour * * * /home/indicatic-e1/Desktop/AutoRun/CheckAlpyLogs.sh & echo \$! > /tmp/CheckAlpyLogs.pid
$BegingmainTESSMinute $BegingmainTESSHour * * * /home/indicatic-e1/Desktop/code/mainTess.sh
$BegingAlpyLogRefreshMinute $BegingAlpyLogRefreshHour * * * /home/indicatic-e1/Desktop/AutoRun/AlpyLogRefresh.sh
$BegingInteropLogRefreshMinute $BegingInteropLogRefreshHour * * * /home/indicatic-e1/Desktop/AutoRun/InteropLogRefresh.sh
$BegingConverterLogRefreshMinute $BegingConverterLogRefreshHour * * * /home/indicatic-e1/Desktop/AutoRun/ConverterLogRefresh.sh
$BegingTESSLogRefreshMinute $BegingTESSLogRefreshHour * * * /home/indicatic-e1/Desktop/AutoRun/TESSLogRefresh.sh
$BegingNikonLogRefreshMinute $BegingNikonLogRefreshHour * * * /home/indicatic-e1/Desktop/AutoRun/NikonLogRefresh.sh
$BegingVerifInternetMinute $BegingVerifInternetHour * * * /home/indicatic-e1/Desktop/AutoRun/VerifInternet.sh
$BegingVerifMemoryMinute $BegingVerifMemoryHour  * * * /home/indicatic-e1/Desktop/AutoRun/VerifFillingMemory.sh

$KillCodeQHYMinute $KillCodeQHYHour * * * pkill -SIGUSR1 -f /home/indicatic-e1/Desktop/INDIcode/qhy
$KillCodeAlpyMinute $KillCodeAlpyHour * * * pkill -SIGUSR1 -f /home/indicatic-e1/Desktop/INDIcode/alpy
$KillCodeNikonMinute $KillCodeNikonHour * * * pkill -SIGUSR1 -f /home/indicatic-e1/Desktop/code/nikon.sh
$KillConverterPYMinute $KillConverterPYHour * * * pkill -SIGUSR1 -f /home/indicatic-e1/Desktop/code/converter.py
$KillCodeTESSMinute $KillCodeTESSHour * * * pkill -SIGUSR1 -f /home/indicatic-e1/Desktop/code/Interop_code/TESS.sh
$KillCheckAlpyLogsMinute $KillCheckAlpyLogsHour * * * pkill -SIGUSR1 -f /home/indicatic-e1/Desktop/AutoRun/CheckAlpyLogs.sh
$KillmainTESSMinute $KillmainTESSHour * * * /home/indicatic-e1/Desktop/code/mainTess.sh
$KillAlpyLogRefreshMinute $KillAlpyLogRefreshHour * * * pkill -SIGUSR1 -f /home/indicatic-e1/Desktop/AutoRun/AlpyLogRefresh.sh
$KillInteropLogRefreshMinute $KillInteropLogRefreshHour * * * /home/indicatic-e1/Desktop/AutoRun/VerifEndSendingFiles.sh
$KillConverterLogRefreshMinute $KillConverterLogRefreshHour * * * pkill -SIGUSR1 -f /home/indicatic-e1/Desktop/AutoRun/ConverterLogRefresh.sh
$KillTESSLogRefreshMinute $KillTESSLogRefreshHour * * * pkill -SIGUSR1 -f /home/indicatic-e1/Desktop/AutoRun/TESSLogRefresh.sh
$KillNikonLogRefreshMinute $KillNikonLogRefreshHour * * * pkill -SIGUSR1 -f /home/indicatic-e1/Desktop/AutoRun/NikonLogRefresh.sh
$KillVerifMemoryMinute $KillVerifMemoryHour * * * pkill -SIGUSR1 -f /home/indicatic-e1/Desktop/AutoRun/VerifFillingMemory.sh
```

Figure 38 – *Organisation des tâches dans Crontab*

12.2 Contrôle de la station

À ce jour, plusieurs scénarios d'urgence potentiels ont été anticipés, tels qu'une éventuelle coupure d'accès à Internet ou un remplissage excessif de la mémoire. Voici comment ils sont traités.

— **Cas coupure d'accès à Internet :** En cas de perte de connexion Internet sur la station, les programmes assurant la communication avec les équipements ainsi que le module d'interopérabilité sont automatiquement arrêtés. Quand la connexion Internet est de nouveau disponible, il y a deux différentes options pour reprendre :

- **Option 1 :** La connexion est de nouveau disponible et le cycle n'est toujours pas terminé, alors le code d'interopérabilité se remet à fonctionner pour transmettre les données et demander des données aux équipements.
- **Option 2 :** La connexion est de nouveau disponible mais le cycle est terminé, alors le code d'interopérabilité va juste envoyer les données acquises avant la coupure de connexion.

— **Mémoire remplie :** En cas d'un seuil prédéterminé atteint, alors les programmes assurant la communication avec les équipements ainsi que le module d'interopérabilité sont automatiquement arrêtés. Quand l'autre seuil qui détermine le moment à partir duquel reprendre à la normale, il y a deux différentes options pour reprendre :

- **Option 1 :** Le seuil minimal est atteint et le cycle n'est pas terminé, alors la station demande de nouveau des données aux équipements.
- **Option 2 :** Le seuil minimal est atteint mais le cycle est terminé, alors seulement l'envoi des données restantes est envoyé.

De plus, même une fois le cycle terminé, le code continue d'envoyer les données tant qu'il en reste, et ne s'interrompt que lorsque toutes les données ont été transmises.

13 Transfert de savoir

GitHub est utilisé afin d'effectuer le transfert de savoir de ce qui a été fait pendant cette période de stage. En effet, GitHub permet de manière simple l'envoi de code et la réception de ceci avec des modifications apportées.

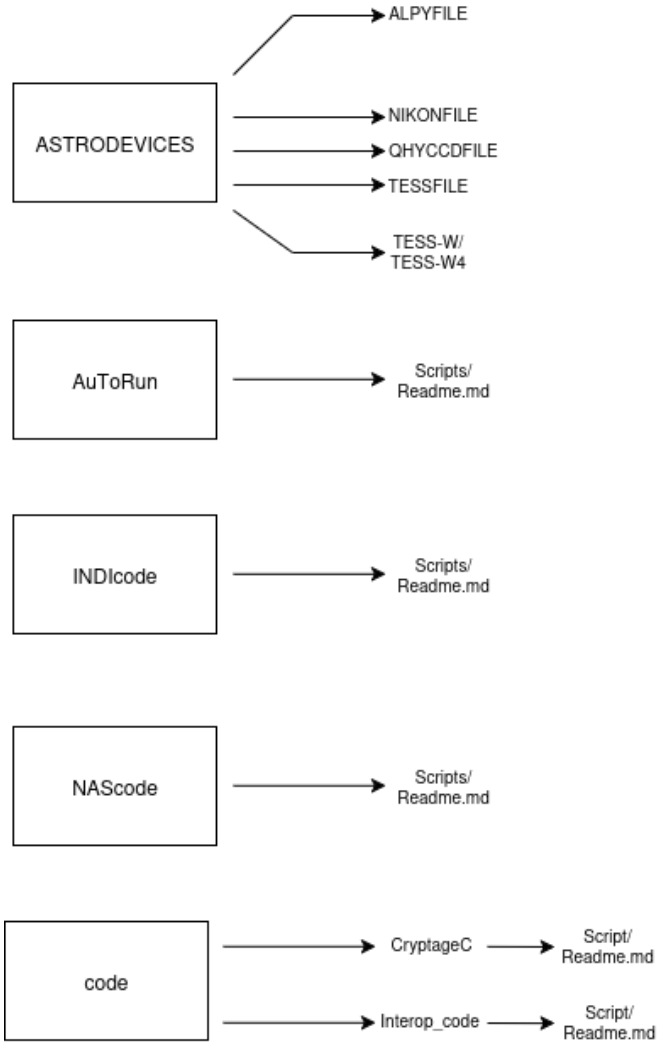

Figure 39 – *Organisation GitHub*

La figure 39 illustre la structure du dépôt GitHub et la répartition des fichiers transférés :

- **ASTRODEVICES :** Les dossiers *ALPYFILE*, *NIKONFILE*, *QHYCCDFILE*, *TESSFILE*, *TESS-W* et *TESS-W4* présents dans le dossier texteASTRODEVICES représentent les répertoires de chaque équipement où sont transférées les données capturées en attente pour leur envoi vers le NAS.
- **AuToRun :** Contient les scripts utilisés pour le fonctionnement de la partie 6.
- **INDIcode :** Contient les scripts utilisés pour communiquer avec le serveur INDI, expliqué dans la section 6.
- **NAScode :** Contient les scripts utilisés pour l'organisation des fichiers reçus par le NAS, ainsi que ceux dédiés au déchiffrement, respectivement présentés dans les parties 4 et 5.
- **code :** Contient les deux dossiers *CryptageC* et *Interop_code*, qui contienent respectivement la partie chiffrements de *Payload* et *Info* (section 8) et ainsi que l'organisation et le traitement des fichiers dans la station (section 7.1).

Il est à noter qu'il est présent dans les dossiers *AuToRun*, *INDIcode*, *NAScode*, *CryptageC* et *Interop_code*, un fichier Readme.md. Dans ces fichiers se trouve l'explication de chaque script ainsi que les paquets à installer pour le bon fonctionnement de chacun. Afin d'avoir un aperçu visuel, une capture d'écran du dépôt GitHub est visualisable sur la figure 40

Figure 40 – *Capture d'écran du dépôt GitHub*

14 Diagramme de Gantt et KPI (Key Performance Indicator)

Cette section du rapport est consacrée à l'organisation des différentes tâches accomplies au cours de la période de stage. Une explication détaillée du diagramme de Gantt est fournie. De plus, les indicateurs de performance, illustrés par un schéma de répartition temporelle et une représentation barométrique du taux de réussite, offrent une vue d'ensemble complète du code développé. Ces éléments permettent de mettre en lumière les points clés du projet, et ainsi d'aboutir à une conclusion pertinente sur sa gestion globale.

14.1 Explication du diagramme de gantt et point de vue sur l'organisation du temps

Le diagramme de Gantt représenté sur la figure 42 permet de voir le temps passé sur chaque tâche des six différentes parties ainsi que le temps fallu pour réaliser la rédaction de ce rapport ainsi que le poster demandé. Voici comment il doit être lu, sur la gauche se situent deux colonnes, une en vert et une en bleu, dans la colonne en vert est indiqué le temps estimé en heures pour chaque tâche tandis que la colonne bleue indique le temps réellement passé. En haut, dans les couleurs bleu clair, sont indiquées les semaines et les jours de la semaine. Ainsi est représenté sur chaque ligne des différentes tâches les temps estimés pour réaliser les tâches en vert et les temps passés en bleu, il est à noter qu'un carré représente 8 heures normalement mais que, dû au fait que plusieurs tâches ont pu être réalisées en parallèle, alors un carré n'est pas considéré comme faisant 8h dans tous les cas. Il est possible de visualiser à deux reprises des fines lignes rouges qui précèdent le temps estimé, cette ligne représente le nouveau temps estimé pour terminer la tâche si celle-ci n'a pas pu être réalisée dans les temps impartis. Enfin, les temps totaux sont indiqués en bas à gauche du diagramme.

Dans la majorité des cas, les délais estimés ont été respectés. Bien qu'un léger dépassement ait eu lieu à deux reprises, une gestion rigoureuse de l'organisation a permis de rattraper le temps perdu, aboutissant ainsi à une réalisation finale plus rapide que prévue.

14.2 KPI

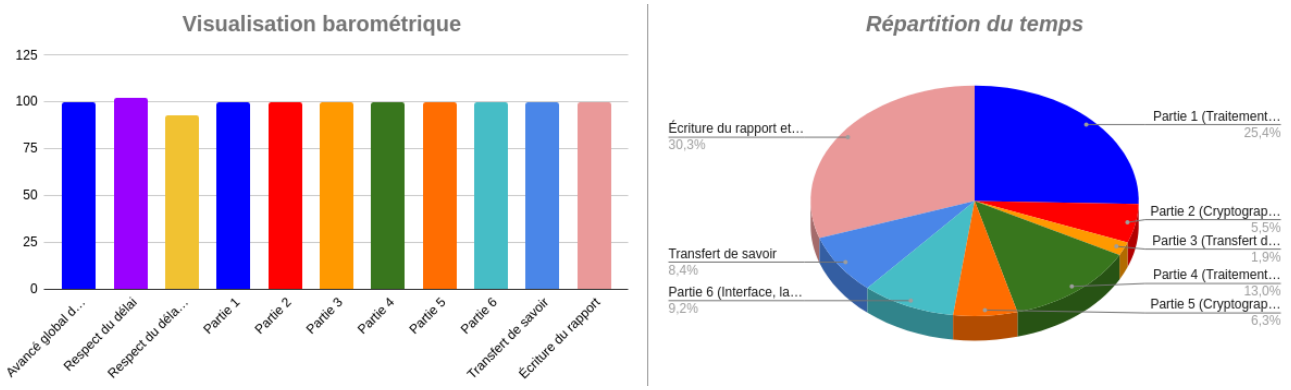

Figure 41 – *KPI*

Le KPI est représenté sous deux formes différentes, sur la droite est indiqué le pourcentage de temps passé sur les six différentes parties du rapport ainsi que le transfert de savoir et l'écriture de ce rapport en plus du poster. La représentation barométrique sur la gauche représente le pourcentage de réussite de chaque partie, ainsi que l'avancée globale des tâches confiées, le respect du délai et le respect du délai de chaque partie. Il apparaît que 38,7 % du temps total a été consacré à la rédaction du rapport et au transfert de savoir. Cela souligne le rôle majeur de la documentation détaillée et du partage d'informations, essentiels pour permettre à d'autres de comprendre le projet et d'assurer sa continuité.

Concernant les différentes sections du code développé, la partie 1 représente 25,4 % du temps total alloué, et 44,3 % du temps cumulé des six parties. Cela reflète l'importance cruciale de la consolidation des fondations du code ainsi que de l'appropriation du projet réalisé.

La représentation barométrique illustre l'avancement des six différentes sections du projet, ainsi que la rédaction du rapport incluant le transfert de savoir. Chaque section ayant une valeur de 100 indique son achèvement complet. Le respect du délai global atteint légèrement plus de 100 %, en raison d'un temps total réellement passé légèrement inférieur à celui initialement estimé (voir diagramme de Gantt). Enfin, le respect des délais pour les différentes tâches (barres en jaune) est de 92,86 %, puisque $\frac{2}{28}$ d'entre elles ont nécessité une révision du temps imparti.

14.3 Conclusion sur la gestion de projet

Il peut ainsi être conclu que l'organisation mise en place durant ce stage a largement contribué à la réussite du projet. En effet, très peu de retards ont été constatés, et grâce au système de réestimation du temps mis en œuvre, les tâches ont pu être achevées sans compromettre le temps final. En effet, 16h de moins ont été réalisées par rapport au temps total estimé. Enfin, le temps consacré au transfert de savoir via GitHub (voir section 13) ainsi qu'à la rédaction du présent rapport garantit la continuité efficace du projet pour les opérations futures.

	Estimé	Passé
Partie 1 (Traitement des données côtés station)		
Partie a : Taitement des données de chaque équipement	128h	128h
Partie b : Fonction execution	24h	22h
Partie c : Fonction Maketram	8h	5h
Partie d : Information de connexion	3h	4h
Partie e : Transfert des données vers le NAS	80h	82h
Partie f : Réalisation des empreintes numériques	1h	1h
Partie 2 (Cryptographie (chiffrement))		
Première étape : Partage de clé secrète de Shamir	10h	14h
Deuxième étape : César généralisé appliqué à une matrice numérique	16h	22h
Troisième étape : Application d'un masque binaire	16h	16h
Partie 3 (Transfert des données)		
Recherche du protocole adéquat et prise en main de son fonctionnement	10h	10h
Prise en main de Tailscale	8h	8h
Partie 4 (Traitement des données côté NAS)		
Partie a : Implémentation de la recherche de fichiers CryptFile existant	32h	32h
Partie b : Récupération des valeurs du fichier CryptFile.txt	20h	20h
Partie d : Organisations des fichiers	96h	72h
Partie 5 (Cryptographie (déchiffrement))		
Première étape : Opération inverse du XOR	12h	12h
Deuxième étape : Opération inverse de César	16h	16h
Troisième étape : Interpolation lagrangienne	32h	32h
Partie 6 (Interface, lancement et contrôle de la station)		
Prise en compte des élements à considérés	24h	24h
Interface et lancement	40h	40h
Contrôle de la station	24h	24h
Transfert de savoir		
Rédaction de la documentation du GitHub	80h	80h
Écriture du rapport et réalisation du poster		
Écriture partie 1	56h	56h
Écriture partie 2	40h	40h
Écriture partie 3	40h	40h
Écriture partie 4	40h	40h
Écriture partie 5	40h	40h
Écriture partie 6	40h	40h
Réalisation du poster	32h	32h
Total de temps passé	**968h**	**952h**

Figure 42 – *Diagramme de Gantt*

15 Conclusion

Le code d'interopérabilité développé a été testé avec succès sur deux stations simultanées pendant une période limitée, démontrant la fiabilité et l'efficacité de chaque étape décrite dans ce rapport. Les résultats obtenus confirment le bon fonctionnement du système, tant au niveau des stations qu'au niveau du NAS, avec une organisation maîtrisée du transfert de données. La confidentialité des échanges a été vérifiée via une analyse des trames sur *Wireshark*, et l'intégrité des données a été validée par l'injection de caractères erronés dans les empreintes numériques de *Payload* et *Info*, générant les réponses attendues. Le débit de transfert dépend principalement de la qualité du réseau auquel chaque station est connectée. Enfin, l'automatisation des stations répond aux exigences fixées, assurant une gestion autonome des cycles ainsi que la prise en charge des cas d'urgence, notamment les coupures d'accès Internet ou le remplissage de la mémoire.

Le transfert de savoir s'est révélé d'une utilité primordiale dans le cadre de ce projet. Il a permis à un nouvel intervenant de s'approprier efficacement le code développé ainsi que la méthode d'installation nécessaire à son bon fonctionnement sur une station. Par ailleurs, la gestion de projet a largement contribué au succès global de l'opération. Elle a permis de coordonner les différentes phases du développement et d'identifier les tâches critiques, sur lesquelles les efforts ont été concentrés afin d'assurer un taux de réussite complet.

15.1 Les perspectives futures du projet

Afin d'éviter toute saturation du NAS en cas d'envoi simultané de données par plusieurs stations, une architecture en réseau maillé est envisagée. Cette solution, rendue possible par l'utilisation de *Tailscale*, permet une interconnexion directe entre les stations. Grâce à ce réseau maillé, les stations peuvent communiquer entre elles et coordonner l'envoi global des données de manière synchronisée. Cela réduit considérablement la sollicitation du processeur du NAS et améliore ainsi le traitement des flux d'information.

Une application est actuellement en cours de développement afin de permettre une visualisation simple et intuitive des différentes stations. Celle-ci affichera diverses informations utiles, comme le nombre total de fichiers envoyés par chaque station, leurs noms, ainsi que la localisation géographique de ces dernières. L'objectif est de faciliter le suivi en temps réel du réseau et d'optimiser la gestion globale du système.

Enfin, si des financements supplémentaires peuvent être alloués au projet, une extension du réseau sera envisagée, portant le nombre total de stations à six. Cette augmentation permettra d'obtenir une couverture plus large et plus représentative du canal de Panama, assurant ainsi une estimation plus précise et significative de l'impact de la pollution sur cette zone stratégique.

16 Références

[1] Adrien.D. (21 Août 2022). *Cron et crontab : le planificateur de tâches.*
URL : `https://www.linuxtricks.fr/wiki/cron-et-crontab-le-planificateur-de-taches`

[2] Alexander V. Lukyanov. (11 Juin 2022). *lftp - Sophisticated file transfer program.*
URL : `https://lftp.yar.ru/lftp-man.html`

[3] Alexandre Slowik, Guillaume Séchet. (1 Juill 2021). *Chaleur : une perception différente selon l'humidité et les individus*
URL : https://www.meteo-paris.com/actualites/chaleur-une-perception-differente-selon-l-humidite-et-les-individus

[4] Astronomie-va. (30 nov 2022). *Le spectre de l'atome d'hydrogène.*
URL : `https://www.astronomie-va.com/forum/viewtopic.php?t=3351`

[5] Astropshop.eu. (s.d). *Camera DSLR D5600a.*
URL : `https://www.astroshop.eu/astromodified-dslr-dslms/nikon-camera-dslr-d5600a/p,53133`

[6] Climats et voyages. (s.d). *Climat - Panama.*
Contenu utilisé : Côté sud.
URL : `https://www.climatsetvoyages.com/climat/panama`

[7] Gregory Kovalchuk. (4 Juill 2024). *Shamir's Secret Sharing : A Step-by-Step Guide with Python Implementation.*
URL : https://medium.com/@goldengrisha/shamirs-secret-sharing-a-step-by-step-guide-with-python-implementation-da25ae241c5d

[8] Moty Michaely. (Avr 2025). *SFTP vs. FTPS benchmarks : file transfer speed comparison 2025.*
URL : `https://sftptogo.com/blog/sftp-vs-ftps-benchmarks/`

[9] Pierrelm. (10 Avr 2025). *Interpolation lagrangienne.*
URL : `https://fr.wikipedia.org/wiki/Interpolation_lagrangienne`

[10] Robert Dougherty. (5 Oct 2023). *Sécurité SFTP – Est-elle vraiment sécurisée ?*
URL : https://www.kiteworks.com/fr/transfert-de-fichier-securise/securite-sftp-est-elle-vraiment-securisee/#Comment_fonctionne_lSFTP

[11] STARS4ALL. (s.d.). *STARS4ALL Foundation.*
URL : `https://foundation.stars4all.eu/`

[12] Tailscale Inc. (s.d.). *Site officiel de Tailscale* : `https://tailscale.com`

[13] DATASHEET DiskStation DS1821+. (2021). *High-capacity storage and data protection for anyone.*
URL : `https://global.download.synology.com/download/Document/Hardware/DataSheet/DiskStation/21-year/DS1821%2B/enu/Synology_DS1821%2B_Data_Sheet_enu.pdf?utm_source`

[14] Lyle Smith. (11 Mars 2021). *Synology DiskStation DS1821+ Review.*
URL : `https://www.storagereview.com/review/synology-diskstation-ds1821-review?utm_source`