Spécifications

AYOUB Pierre, BASKEVITCH Claire, BESSAC Tristan,
CAUMES Clément, DELAUNAY Damien, DOUDOUH Yassin

Mercredi 18 Avril 2018

StegX

Table des matières

1 Introduction

En réalisant le cahier des charges concernant l'application StegX, l'analyse des contraintes et des besoins du client nous amène à rédiger les spécifications.

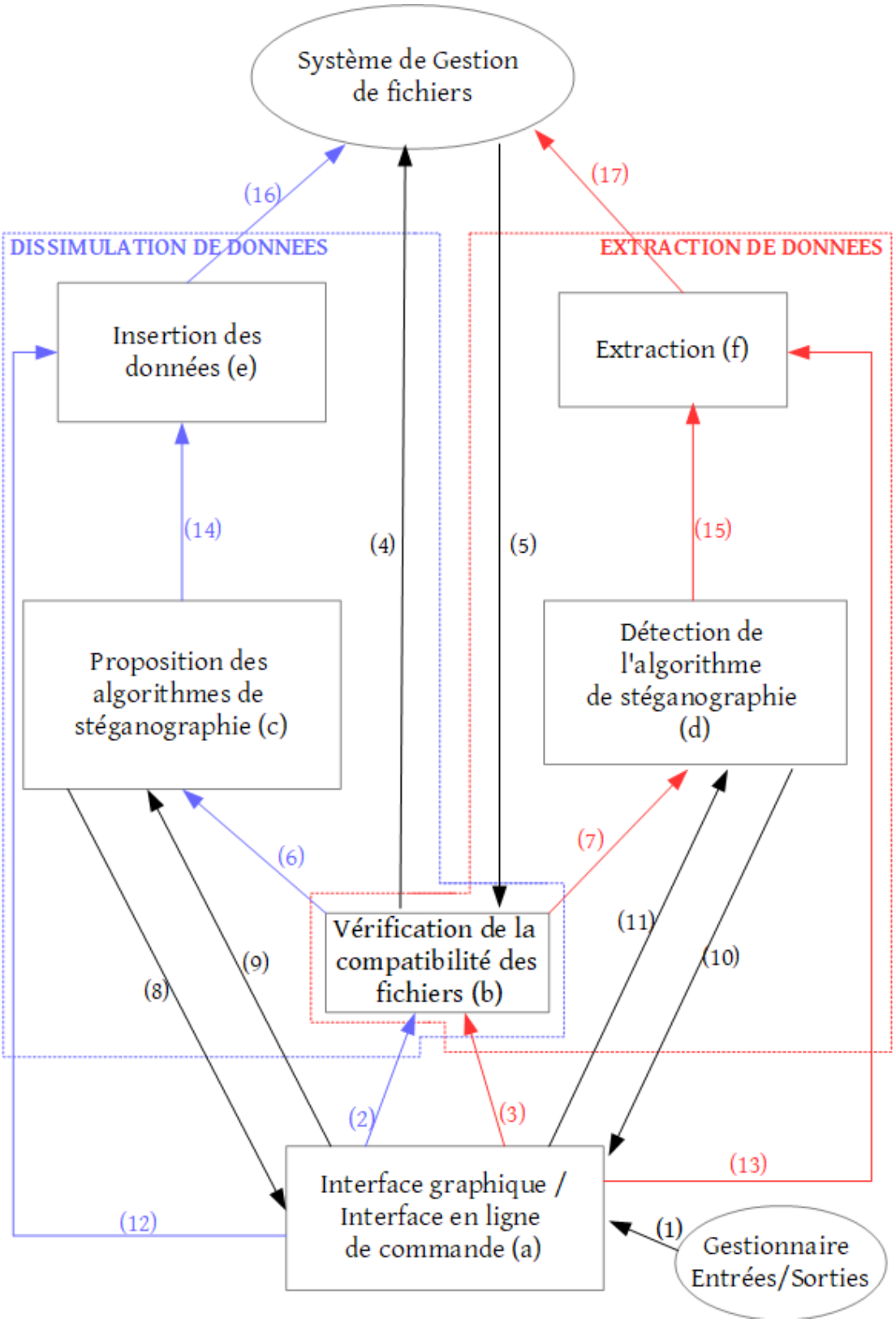
Le logiciel StegX pourra proposer de cacher des données dans des fichiers dont le format est pris en charge par l'application. De plus, un utilisateur pourra extraire les données cachées d'un fichier qu'on considère déjà comme hôte. StegX prendra en charge les trois algorithmes de stéganographie suivants : EOF, LSB et MetaData.

Avec l'analyse des fonctionnalités de l'application, nous avons vu qu'elle devait gérer la manipulation de données binaires et donc il fallait un langage proche de la machine. De plus, nous devons implémenter des fonctions de stéganographie afin de cacher des données dans des fichiers. Par conséquent, une programmation procédurale était nécessaire. Enfin, il fallait des structures ou des objets simples afin de manipuler des structures de fichiers. De ce fait, nous avons choisi le langage C pour implémenter la future application StegX. En effet, le langage C propose une programmation procédurale, ainsi qu'un langage bas niveau proche de la machine. Enfin, la possibilité d'écriture et de lecture de bits et d'accès mémoire avec le langage C correspond à nos besoins. C'est donc pour toutes ces raisons que l'équipe de conception de StegX a choisi le langage C pour implémenter l'application.

Après l'étude des différents modules de l'organigramme, ainsi que les informations qui se déplacent, nous avons établi un organigramme. Cet organigramme nous permettra ainsi de rédiger en détail les spécifications des différents modules de l'application.

2 Modules du produit

2.1 Organigramme



2.2 Liste des modules et de leurs fonctionnalités

a) Interface graphique / Interface en ligne de commande : interfaces permettant à l'utilisateur de choisir parmi les deux fonctionnalités possibles de l'application. Il peut dissimuler des données dans un fichier (dont le type et le format sont pris en charge par l'application). Ou bien, il peut extraire les données cachées d'un fichier.
Il aura donc un mécanisme pour choisir le fichier hôte et le fichier à cacher (pour la dissimulation de données), et un mécanisme pour choisir le fichier contenant les données cachées à analyser (pour l'extraction de données cachées), grâce à une interaction avec le gestionnaire Entrée/Sorties.

b) Vérification de la compatibilité des fichiers : le format du fichier hôte (pour le module *Dissimulation de données*) ou le format du fichier à analyser (pour le module *Extraction de données cachées*), choisis par l'utilisateur, est vérifié pour savoir s'il est bien pris en charge par l'application.
Il aura un mécanisme de lecture du fichier hôte : selon les formats pris en charge, il y aura une batterie de tests pour déterminer le format du fichier. Il prendra en entrée les chemins de fichiers en entrée. Ce module interagira avec le gestionnaire de Gestion de Fichiers.

c) Proposition des algorithmes de stéganographie : en fonction du type et du format du fichier hôte, ainsi que de la taille des données à cacher, un mécanisme proposera un ou plusieurs algorithmes (avec ou sans sécurité supplémentaire). Si il y a un mot de passe, l'interface va donner le mot de passe à ce module, ainsi que l'algorithme choisi.

d) Détection de l'algorithme de stéganographie : un mécanisme d'analyse du fichier permettra de découvrir quel algorithme a été utilisé afin de les extraire correctement par la suite. S'il est nécessaire d'utiliser un mot de passe pour déchiffrer les données cachées, le module récupèrera le champ mot de passe de l'interface.

e) Insertion des données : la copie des données du fichier hôte sera modifiée avec l'insertion des données à cacher à l'aide de l'algorithme choisi par l'utilisateur. Il prendra en entrée le fichier à cacher, le fichier hôte, le chemin du fichier résultant, l'algorithme à utiliser ainsi que le mot de passe s'il y en a un.

f) Extraction : les données cachées dans le fichier à analyser sont extraites. Les algorithmes utilisés pour l'insertion des données et l'extraction seront expliqués dans la sous-section *"Algorithmes de dissimulation et d'extraction en stéganographie"*. Ce module prendra en entrée le fichier à analyser, le chemin du fichier à extraire et l'algorithme détecté.

3 Description des structures de données

3.1 Description des énumérations

```
enum mode {STEGX_MODE_INSERT, STEGX_MODE_EXTRACT};
typedef enum mode mode_e;
```

L'énumération mode permet de distinguer les deux outils proposés par l'application : extraire ou dissimuler.

```
enum algo {STEGX_ALGO_LSB, STEGX_ALGO_EOF, STEGX_ALGO_METADATA};
typedef enum algo algo_e;
```

L'énumération algo distingue les différents algorithmes proposés par l'application.

```
enum type {BMP_COMPRESSED, BMP_UNCOMPRESSED, PNG, WAV_PCM, WAV_NO_PCM,
        MP3, AVI_COMPRESSED, AVI_UNCOMPRESSED, FLV, UNKNOWN};
typedef enum type type_e;
```

L'énumération type expose les différents formats de fichiers avec leurs particularités pour certains. Si le format du fichier hôte ou à analyser est inconnu, cela sera représenté par le type UNKNOWN.

3.2 Description des types

3.3 Description des structures

```
struct stegx_info_ins {
    char* hidden_path;
    algo_e algo;
};
typedef struct stegx_info_ins stegx_info_ins_s;
```

La structure stegx_info_ins permet de représenter les informations nécessaire sur le fichier à cacher, c'est-à-dire son chemin et l'algo que l'on utilisera pour dissimuler ce dernier.
Paramètres :

- *hidden_path* est une chaine de caractères représentant le nom du fichier à cacher.
- *algo* représente l'algorithme qui sera utilisé pour dissimuler.

```
struct stegx_choices {
    char* host_path;
    char* res_path;
    char* passwd;
    mode_e mode;
    stegx_info_ins_s* ins_info;
};
typedef struct stegx_choices stegx_choices_s;
```

La structure stegx_choices représente la structure publique (de la bibliothèque de StegX). Elle représente les données choisies par l'utilisateur s'il veut dissimuler ou extraire des données.
Paramètres :

— *host_path* est une chaine de caractères représentant le chemin du fichier à analyser (pour l'extraction) ou hôte (pour la dissimulation).
— *res_path* est une chaine de caractères représentant le chemin du fichier résultant (pour l'extraction ou pour la dissimulation).
— *passwd* est une chaine de caractères représentant le mot de passe choisi par l'utilisateur. Celui-ci est optionnel et si l'utilisateur choisit de ne pas en mettre, passwd vaudra NULL.
— *mode* est une variable d'énumération représentant le mode que l'utilisateur a choisi : dissimulation (STEGX_MODE_INSERT)ou extraction (STEGX_MODE_EXTRACT).
— ** ins_info* est un pointeur sur la structure stockant les informations sur le fichier à cacher. Elle est requise si le champ mode est à STEGX_MODE_INSERT.

```
struct host_info {
    FILE* host;
    type_e type;
    union {
        struct bmp bmp;
        struct png png;
        struct wav wav;
        struct mp3 mp3;
        struct avi avi;
        struct flv flv;
    } file_type;
};
typedef struct host_info host_info_s;
```

Cette structure host_info permet de stocker les informations utiles pour chaque format différent afin de le manipuler convenablement par la suite.
Paramètres :
— *host* représente le fichier hôte pour le cas de la dissimulation et le fichier à analyser pour le cas de l'extraction.
— *type* permet de distinguer quel champ il faut manipuler pour l'union infos. Pour chaque format, il y un ou plusieurs types possibles.
— *infos* est une union en C. Elle permet de choisir un champ parmi ceux qui sont définis. Ainsi, si le fichier hôte est un fichier BMP par exemple, seule la structure bmp de infos sera initialisée et manipulée par la suite.

```
struct info {
    mode_e mode;
    host_info_s host;
    FILE* res;
    FILE* hidden;
    char* hidden_name;
    int hidden_length;
    char* passwd;
    algo_e algo;
};
typedef struct info info_s;
```

La structure info correspond à la structure privée de la bibliothèque. Elle va contenir toutes les informations utiles pour la dissimulation ou l'extraction des données.
Paramètres :

— *mode* est une variable d'énumération représentant le mode que l'utilisateur a choisi : dissimulation (STEGX_MODE_INSERT)ou extraction (STEGX_MODE_EXTRACT).
— *host* va contenir le détail du fichier hôte pour le bon déroulement de la dissimulation/extraction (selon les cas d'utilisation).
— *res* représente le fichier résultat qui va être créé pour la dissimulation ou l'extraction.
— *hidden* représente le fichier à cacher dans le cas de l'insertion. Il est donc requis si mode vaut STEGX_MODE_INSERT.
— *hidden_name* représente le nom du fichier à cacher/caché selon les cas.
— *hidden_length* représente la taille du fichier à cacher/caché. Cette information est nécessaire dans l'extraction pour connaitre combien de données il reste à extraire dans le fichier. Dans la dissimulation, ce champ permet de construire correctement les informations globales du fichier caché.
— *passwd* est la chaine de caractères représentant le mot de passe choisi par l'utilisateur.
— *algo* est l'énumération représentant l'algorithme qui sera utilisé pour la dissimulation ou l'extraction.

4 Description des différents modules

4.1 Interface graphique

L'interface graphique permettra à un utilisateur de manipuler correctement l'application StegX. Ainsi, un utilisateur non initié à l'informatique pourra facilement utiliser notre l'application.

Dans le cadre du développement de notre logiciel, nous privilégions les solutions multiplate-formes et libres. L'idéal serait donc une bibliothèque graphique remplissant ces critères, en plus d'être relativement haut niveau et utilisable en C. De nombreuses bibliothèques graphiques proposent de tels fonctionnalités, par exemple Qt, GTK+, wxWidgets pour ne citer qu'eux. Lors de notre précédent projet, nous avions déjà appris à utiliser Qt. Nous souhaitions dès lors apprendre à utiliser une nouvelle bibliothèque plutôt que nous baser sur nos acquis, nous avons donc opté pour GTK+ qui correspondait à tout notre critères.

GTK+ est une bibliothèque graphique libre, multiplate-forme, développée à l'origine pour le logiciel de traitement d'image GIMP. Très utilisée dans l'environnent de bureau GNOME, cette bibliothèque propose un large panel de fonctionnalité, dont le paradigme des widgets pour la création de l'interface et celui des signaux pour la gestion de l'interaction avec l'utilisateur. Travailler avec GTK+ nous permettra de nous familiariser avec l'environnent et les bibliothèques utilisés par GNOME et GTK+, à savoir GDK, GLib, GIO et GObject.

Pour écrire ces spécifications, nous avons donc eu à étudier quels widgets seront utilisés, ainsi que les signaux à mettre en oeuvre pour mettre en relation ces derniers avec les actions de l'utilisateur. Dans le but de stocker les informations de l'interface utilisateur, nous allons créer une structure nommée *struct ui* qui contiendra tous les widgets utilisés dans l'interface ainsi que les différents messages affichés par le programme.

```
void ui_create (GtkWidget* window, struct ui* ui);
```

Cette fonction va permettre de créer entièrement l'interface utilisateur sur une fenêtre donnée. Elle construira les widgets et configure les signaux.
Entrées :
— **window :* pointeur vers la fenêtre sur laquelle construire l'interface utilisateur.
— **ui :* pointeur vers l'interface utilisateur à remplir.

```
void ui_delete (struct ui* ui);
```

Cette fonction va supprimer l'interface utilisateur. Elle permettra de désallouer la mémoire utilisée pour l'interface utilisateur.

Entrée :
— **ui :* pointeur vers l'interface utilisateur à désallouer.

```
void ui_build (struct ui* ui);
```

Cette fonction va construire la fenêtre principale en ajoutant les conteneurs et les widgets.
Entrée :
— **ui :* pointeur vers l'interface utilisateur où il faut construire l'affichage.

```
void ui_signal_init (struct ui* ui);
```

Cette fonction va initialiser les signaux et connecter ces derniers aux widgets de la fenêtre principale.
Entrée :
— **ui :* pointeur vers l'interface utilisateur où il faut configurer les signaux.

La structure *struct ui* et les 4 fonctions sont les outils principaux pour construire l'interface graphique. Ainsi, cela permet de visualiser en amont le fonctionnement de l'interface graphique de StegX.

4.2 Interface en ligne de commande

Concernant l'interface en ligne de commande, c'est le deuxième moyen pour manipuler l'application StegX. Les utilisateurs savant manipuler le terminal peuvent ainsi faire de la stéganographie.

```
stegx_choices_s* init_stegx_choices ();
```

Cette fonction va initialiser la structure contenant les informations entrées en ligne de commande.
Sortie :
— *stegx_info_s :* renvoie la structure initialisée.

```
void fill_info (stegx_choices_s* com, const int argc, char* const* argv);
```

La fonction va remplir la structure avec les informations entrées en ligne de commande.
Entrée :
— **com :* pointeur sur la structure contenant les informations entrées par l'utilisateur.
— *argc :* nombre d'arguments entrés en ligne de commande.
— *argv :* arguments entrés en ligne de commande.

```
void we_are_stegx ();
```

Cette fonction va afficher la présentation du projet.

```
void help ();
```

La fonction *help* va afficher l'aide du lancement de l'application en ligne de commande et détaille les différents paramètres à spécifier pour l'utilisateur.

```
void unvalid_line (stegx_choices_s* com);
```

Cette fonction avertira l'utilisateur d'une erreur lors du lancement de l'application en ligne de commande et affiche l'aide.

Entrée :

— **com :* pointeur sur la structure contenant les informations entrées par l'utilisateur.

```
void check_info (stegx_choices_s* com);
```

Cette fonction vérifiera les informations entrées par l'utilisateur et que l'utilisateur a bien indiqué les informations nécessaires pour la dissimulation ou l'extraction.

Entrée :
— **com :* pointeur sur la structure contenant les informations entrées par l'utilisateur.

```
void dest_stegx_choices (stegx_choices_s* com);
```

Cette fonction va libérer la structure contenant les informations entrées en ligne de commande.

Entrée :
— **com :* pointeur sur la structure contenant les informations entrées par l'utilisateur.

4.3 Vérification de la compatibilité des fichiers

D'après l'organigramme créé lors du cahier des charges, ce module prend en entrée les chemins du fichier hôte et du fichier à cacher (pour la dissimulation) ; le chemin du fichier à analyser (pour l'extraction). Il va interagir avec le système de fichiers pour pouvoir vérifier la compatibilité des fichiers en entrée. Enfin, après sa vérification, ce module va envoyer les fichiers aux modules *Proposition des algorithmes de stéganographie* (pour dissimuler) ou *Détection de l'algorithme de stéganographie* (pour extraire). Ainsi, cette communication se fera à l'aide de la structure privée de la bibliothèque *info_s*.

```
info_s* stegx_check_compatibility_dissimulation (stegx_choices_s* infos);
```

Cette fonction analyse la compatibilité des fichiers hôte et à cacher pour la dissimulation. Ces données sont dans la structure info mis en paramètre. Pour cela, elle vérifie que les fichier hôte et à cacher existent et qu'ils ne sont pas vides. Ensuite, la fonction vérifie le format du fichier hôte. Elle va remplir les champs *mode*, *passwd*, *hidden_name*, *hidden*, *res* et *host* (partiellement) de la structure *info_s** en sortie.
Entrée :
— **infos :* structure publique que les interfaces vont initialiser.
Sortie :
— *info_s* :* pointeur sur une structure initialisée partiellement pour insérer les informations nécessaires à la vérification de la compatibilité des fichiers en entrée.

```
info_s* stegx_check_compatibility_extraction (stegx_choices_s* infos);
```

Cette fonction analyse la compatibilité du fichier à analyser pour l'extraction. Pour cela, elle vérifie si le fichier à analyser existe et n'est pas vide. Ensuite, la fonction vérifie le format du fichier à analyser. Elle va remplir les champs *mode*, *hidden_name*, *hidden*, *res* et *host* (partiellement) de la structure *info_s** en sortie.
Entrée :
— **infos :* structure publique que les interfaces vont initialiser.
Sortie :
— *info_s* :* pointeur sur une structure initialisée partiellement pour insérer les informations nécessaires à la vérification de la compatibilité des fichiers en entrée.

```
int stegx_test_file_not_empty (FILE* file);
```

Cette fonction teste si le fichier n'est pas vide.
Entrée :
— *file :* fichier à tester.
Sortie :
— *int :* renvoie 0 si le fichier est vide ou qu'il n'existe pas ; 1 sinon.

```
type_e stegx_test_file_bmp (FILE* file);
type_e stegx_test_file_png (FILE* file);
type_e stegx_test_file_wav (FILE* file);
type_e stegx_test_file_mp3 (FILE* file);
type_e stegx_test_file_avi (FILE* file);
type_e stegx_test_file_flv (FILE* file);
```

Ces fonctions testent le format du fichier en lisant le fichier en entrée (Magic Number notamment).
Entrée :
— *file :* fichier à tester.
Sortie :
— *type_e :* renvoie UNKNOWN si le format du fichier n'est pas pris en charge par l'application ; sinon, en fonction du format et des particularités du fichier, il peut renvoyer *BMP_COMPRESSED*, *BMP_UNCOMPRESSED*, *PNG*, *WAV_PCM*, *MP3*, *AVI_COMPRESSED*, *AVI_UNCOMPRESSED* ou *FLV*.

```
type_e check_file_format (FILE* file);
```

Cette fonction récupère le fichier dont le format est à trouver. Elle établit une batterie de tests pour déterminer si le format du fichier est pris en charge par l'application. Pour cela, on utilisera les fonctions $type\_e\ stegx\_test\_file\_XXX(FILE * file)$ comme vu précedemment.
Entrée :
— *file :* fichier à tester.
Sortie :
— *type_e :* renvoie UNKNOWN si le format du fichier n'est pas pris en charge par l'application ; sinon, en fonction du format et des particularités du fichier, il peut renvoyer *BMP_COMPRESSED*, *BMP_UNCOMPRESSED*, *PNG*, *WAV_PCM*, *MP3*, *AVI_COMPRESSED*, *AVI_UNCOMPRESSED* ou *FLV*.

4.4 Proposition des algorithmes de stéganographie

Dans l'organigramme des modules de StegX, nous avons créé un sous-module *Proposition des algorithmes de stéganographie*. Il prend en entrée le fichier hôte, le fichier à cacher, le choix de l'utilisateur pour l'algorithme ainsi que le mot de passe voulu par l'utilisateur. Ce sous-module va donc ajouter des données à la structure *info_s* avec notamment l'algorithme de stéganographie utilisé choisi à partir de ceux proposés.

```
void* stegx_suggest_algo_stegano (info_s* infos, stegx_choices_s* user_choices);
```

Cette fonction va remplir les champs *passwd* et *host.file_type* de *info* pour le sous-module suivant *insertion des données*. De plus, elle va changer les valeurs du tableau représentant les algorithmes de stéganographie utilisés. Ainsi, à partir de ce tableau, l'interface graphique proposera certains algorithmes, et l'interface en ligne de commande comparera ce tableau avec le choix de l'utilisateur.
Entrées :

— *infos :* structure représentant toutes les informations pour réaliser l'insertion par la suite.
— *user_choices :* structure représentant les choix de l'utilisateur pour les deux interfaces.

Sortie :

— *void* :* renvoie NULL si tout se passe bien. [a remodifier comme definition].

```
int fill_host_file_type (info_s* infos);
```

En fonction du champ *infos.host.type* représentant le format du fichier hôte, cette fonction va remplir le champ *host.file_type* de *infos*. En effet, *file_type* est une union et est particulière en fonction du format du fichier hôte.

Entrée :

— *infos :* structure représentant toutes les informations pour l'insertion.

Sortie :

— *int :* renvoie 0 si il y a une erreur détectée dans cette fonction et 1 si tout se passe bien.

```
int can_use_algo_lsb (info_s* infos);
int can_use_algo_eof (info_s* infos);
int can_use_algo_metadata (info_s* infos);
```

Ces fonctions vont permettre de savoir si les différents algorithmes (LSB, EOF, Metadata) peuvent être utilisés en fonction des entrées de l'utilisateur à travers l'interface (graphique ou en ligne de commande).

Entrée :

— *infos :* structure représentant toutes les informations pour l'insertion.

Sortie :

— *int :* renvoie 0 si le fichier hôte ne peut pas cacher des données selon l'algorithme, et 1 s'il le peut.

```
int stegx_choose_algo_stegano (info_s* infos, stegx_choices_s* user_choices);
```

A partir du champ *ins_info.algo* de la structure *user_choices*, ainsi que du tableau représentant les algorithmes utilisables (selon les entrées de l'utilisateur), cette fonction change le champ *algo* de *infos*.

Entrées :

— *infos :* structure représentant toutes les informations pour l'insertion.
— *user_choices :* structure représentant les choix de l'utilisateur pour les deux interfaces.

Sortie :

— *int :* renvoie 0 si il y a une erreur détectée dans cette fonction et 1 si tout se passe bien.

4.5 Détection de l'algorithme de stéganographie

Le sous-module *Détection de l'algorithme de stéganographie* prend en entrée le fichier à analyser. De plus, il récupère grâce à l'interface le mot de passe choisi par l'utilisateur. Ce sous-module va compléter les champs des structures *info_s* et *stegx_info_s* avec en particulier l'algorithme de stéganographie utilisé par l'émetteur et que l'application doit détecter pour extraire correctement les données cachées.

```
int stegx_read_signature_app (info_s* infos, stegx_choices_s user_choices);
```

Cette fonction va lire la signature insérée dans le fichier à analyser. Cette signature est constituée d'un octet représentant quel algorithme a été utilisé, avec le nom et la taille du fichier caché.

Entrées :

— *infos :* structure représentant toutes les informations pour l'extraction.
— *user_choices :* structure représentant les choix de l'utilisateur.

Sortie :

— *int :* renvoie 0 si il y a une erreur détectée dans cette fonction et 1 si tout se passe bien.

```
void* detect_algo_stegano (info_s* infos, stegx_choices_s* user_choices);
```

Cette fonction va tout d'abord récupérer le mot de passe choisi par l'utilisateur et initialisé le champ *infos.passwd*. Ensuite, à l'aide de la fonction *fill_host_file_type*, le champ-union *infos.host.file_type* sera initialisé. Enfin, cette fonction appellera *stegx_read_signature_app* pour connaître les informations concernant le fichier caché.

Entrées :

— *infos :* structure représentant toutes les informations pour l'extraction.

— *user_choices :* structure représentant les choix de l'utilisateur.

Sortie :

— *void* :* renvoie NULL si tout se passe bien. [a remodifier comme definition].

4.6 Insertion des données

4.7 Extraction

5 Bilan des informations circulantes entre les modules

6 Conclusion

Pour chaque module et sous-module de l'application, nous avons questionné nos besoins et réfléchi comment répondre à ces derniers. Pour cela, il a fallu réfléchir, à l'aide de l'organigramme créé pour le cahier des charges, sur les signatures des fonctions ainsi que sur les informations qui circulent entre les différents modules. La partie la plus délicate avec la réflexion des structures de données est primordiale pour que les informations circulantes restent cohérentes. En effet, il était nécessaire de respecter le déplacement des données entre les modules selon l'organigramme.

Maintenant que les besoins ont été identifiés (dans le cahier des charges) et que les structures de données et signatures des fonctions ont été analysées (dans les spécifications), il est maintenant nécessaire d'implémenter StegX.