# $S^3$ - Safe Security System

Bellone Sylvain - Mao Julien - Mérault Dimitri - Taront Vincent

Juin 2005

# Table des matières

# 1 Introduction

Dans ce rapport, nous allons vous présenter $S^3$ (Safe Security System), un logiciel exceptionnel offrant à la fois des possibilités de cryptage de données, de compression de fichiers, exportation de fichiers textes (.txt) en fichiers Latex, PDF et qui permet également de cacher un document à l'intérieur d'une image.
La simplicité d'utilisation, l'interface conviviale et les nombreux modules complets en feront un logiciel populaire. La fonction de chiffrement garantie l'intégrité des données d'un utilisateur $S^3$ utilise l'algorithme Rijndael (prononcé "Raindal") plus communément appelé AES pour Advanced Encryption Standard. AES est L'algorithme de chiffrement destiné à remplacer DES (Data Encryption Standard) qui est devenu trop faible au regard des attaques actuelles. Ce cahier détaillera les différents modules de chiffrement, compression, stéganographie et exportation en fichiers Latex et PDF.

# 2 Le chiffrement (Dimitri et Vincent)

## 2.1 Coupage en blocs de données. (Julien)

Le coupage de données est une fonction primordiale qui permet de couper les fichiers texte en blocs de données dont la taille est fixe, il s'agit d'un tableau de 4 lignes et de 4 colonnes de caractère. Les blocs de données, appelés state, subissent un certain nombre de transformation lors des rounds.

**Spécification :** la fonction de coupage prend en paramètre le fichier texte et retourne un pointeur sur l'ensemble des blocs de données, cette fonction fait appelle à une fonction longueur que j'ai implémenté. Elle permet de calculer le nombre de blocs de données nécessaires à allouer en mémoire afin de pouvoir stocker le fichier texte dans ces blocs.

**Principe algorithmique :** la fonction de coupage en blocs de données fait appelle à la fonction longueur afin de pouvoir allouer le nombre de blocs de données : il s'agit d'un tableau à 3 dimensions. Puis on extrait un caractère tant que l'on n'a pas atteint la fin du fichier. Chacun de ces caractères sont stockés dans une case du tableau. Si on a atteint la fin du fichier et que les cases du dernier bloc de données ne sont pas toutes remplies on les remplies avec /0. On retourne un pointeur sur l'ensemble des blocs de données.

## 2.2 Présentation de l'algorithme Rijndael

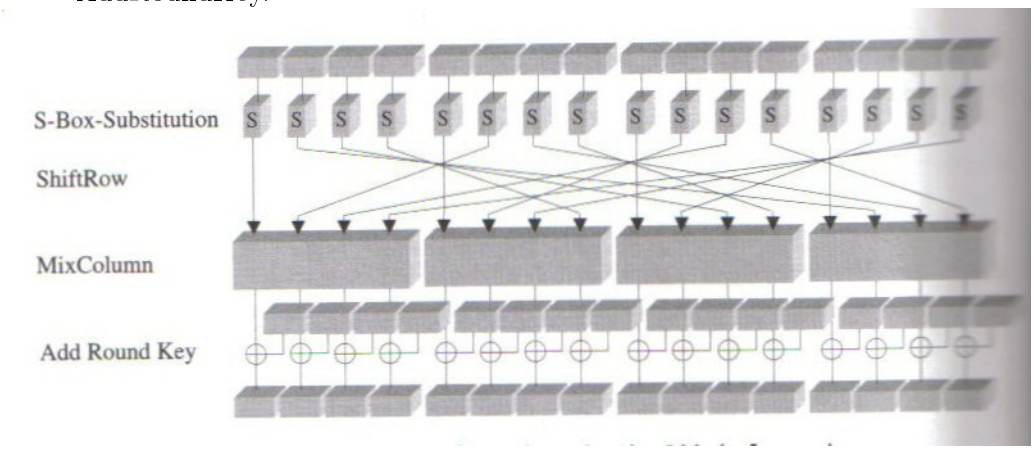
Rijndael est un algorithme de chiffrement dont la taille des blocs de données et la taille des clés peuvent varier. Il peut générer des blocs et des clés de tailles de 128, 192 et 256 bits, toutes les combinaisons de blocs et de clés de tailles différentes sont possibles. Plus tard nommé AES, cet algorithme n'acceptera que des blocs de données de tailles 128 bits bien que la taille des clés puisse toujours varier.

Chaque partie d'un texte est chiffrée plusieurs fois en répétant la même série de transformations. Cette série est essentiellement constituée de 4 transformations et est appelée round :

- SubBytes.
- ShiftRows.
- MixColumn.
- AddRoundKey.

Le dernier round n'est composé que de 3 transformations :

- SubBytes.
- ShiftRows.
- AddRoundKey.



Plus le nombre de round est élevé plus la résistance d'un bloc de données chiffré face à la cryptanalyse est élevée. Le nombre de rounds dépend de la taille des clés.

| | $N_b$ | | | | |
|---|---|---|---|---|---|
| $N_k$ | 4 | 5 | 6 | 7 | 8 |
| 4 | 10 | 11 | 12 | 13 | 14 |
| 5 | 11 | 11 | 12 | 13 | 14 |
| 6 | 12 | 12 | 12 | 13 | 14 |
| 7 | 13 | 13 | 13 | 13 | 14 |
| 8 | 14 | 14 | 14 | 14 | 14 |

## 2.3 SubBytes :

Pour cette tranformation, on s'aidera d'une table de substitution (appelée S-box). Comme son nom l'indique, elle nous permettera de remplacer certaines cases d'indice (x,y) de ce dernier par les cases des blocs de données à chiffrer. Les indices (x,y) sont obtenus à l'aide de la donnée à chiffrer : les 4 bits de poids faible correspondent à l'indice y et les 4 bits de poids fort à l'indice x. Par exemple :


| hex | y | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| x | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
| 0 | 63 | 7c | 77 | 7b | f2 | 6b | 6f | c5 | 30 | 01 | 67 | 2b | fe | d7 | ab | 76 |
| 1 | ca | 82 | c9 | 7d | fa | 59 | 47 | f0 | ad | d4 | a2 | af | 9c | a4 | 72 | c0 |
| 2 | b7 | fd | 93 | 26 | 36 | 3f | f7 | cc | 34 | a5 | e5 | f1 | 71 | d8 | 31 | 15 |
| 3 | 04 | c7 | 23 | c3 | 18 | 96 | 05 | 9a | 07 | 12 | 80 | e2 | eb | 27 | b2 | 75 |
| 4 | 09 | 83 | 2c | 1a | 1b | 6e | 5a | a0 | 52 | 3b | d6 | b3 | 29 | e3 | 2f | 84 |
| 5 | 53 | d1 | 00 | ed | 20 | fc | b1 | 5b | 6a | cb | be | 39 | 4a | 4c | 58 | cf |
| 6 | d0 | ef | aa | fb | 43 | 4d | 33 | 85 | 45 | f9 | 02 | 7f | 50 | 3c | 9f | a8 |
| 7 | 51 | a3 | 40 | 8f | 92 | 9d | 38 | f5 | bc | b6 | da | 21 | 10 | ff | f3 | d2 |
| 8 | cd | 0c | 13 | ec | 5f | 97 | 44 | 17 | c4 | a7 | 7e | 3d | 64 | 5d | 19 | 73 |
| 9 | 60 | 81 | 4f | dc | 22 | 2a | 90 | 88 | 46 | ee | b8 | 14 | de | 5e | 0b | db |
| a | e0 | 32 | 3a | 0a | 49 | 06 | 24 | 5c | c2 | d3 | ac | 62 | 91 | 95 | e4 | 79 |
| b | e7 | c8 | 37 | 6d | 8d | d5 | 4e | a9 | 6c | 56 | f4 | ea | 65 | 7a | ae | 08 |
| c | ba | 78 | 25 | 2e | 1c | a6 | b4 | c6 | e8 | dd | 74 | 1f | 4b | bd | 8b | 8a |
| d | 70 | 3e | b5 | 66 | 48 | 03 | f6 | 0e | 61 | 35 | 57 | b9 | 86 | c1 | 1d | 9e |
| e | e1 | f8 | 98 | 11 | 69 | d9 | 8e | 94 | 9b | 1e | 87 | e9 | ce | 55 | 28 | df |
| f | 8c | a1 | 89 | 0d | bf | e6 | 42 | 68 | 41 | 99 | 2d | 0f | b0 | 54 | bb | 16 |

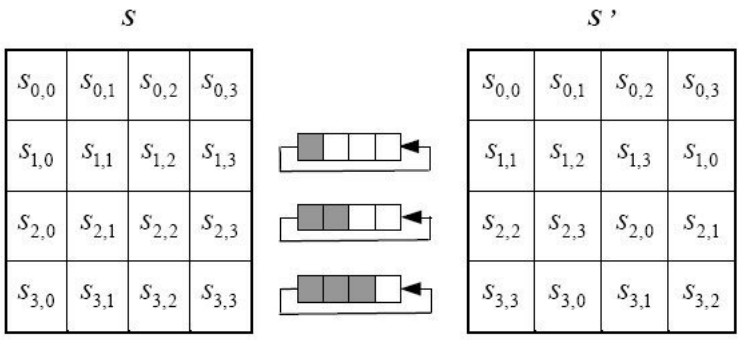
**Spécification :**

La procédure prend en paramètre le tableau contenant les blocs de données a chiffrer et le nombre de bloc.

## 2.4 ShiftRows :

**Principe algorithmique :**

On va procéder à une ou plusieurs permutations des blocs de données sur chaque ligne du tableau contenant les données. Seule la première ligne ne sera pas modifiée. La seconde subira une permutation, la troisième deux permutations, ainsi de suite comme le montre le schéma suivant :




**Spécifiation :**

La procédure `ShiftRows()` prend en paramètre le tableau contenant les blocs de données ainsi que le nombre de bloc.

### 2.5 MixColumns

Voici le schéma des transformations seulement effectuées sur une colonne c :
- t = c[0] XOR c[1] XOR c[2] XOR c[3]
- u = c[0]
- - v = c[0] XOR c[1]
  - v = xtime(v)
  - c[0]=c[0] XOR v XOR t
- - v = c[1] XOR c[2]
  - v = xtime(v)
  - c[1]=c[1] XOR v XOR t
- - v = c[2] XOR c[3]
  - v = xtime(v)
  - c[2]=c[2] XOR v XOR t
- - v = c[3] XOR u
  - v = xtime(v)
  - c[3]=c[3] XOR v XOR t

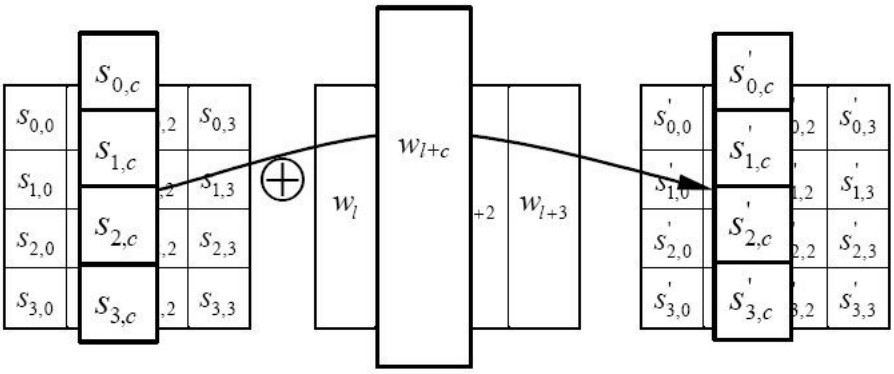
Xtime est similaire a un SubBite :
Soit un nombre hexadécimale codé sur deux caractères : 0a

Xtime(0a) correspondera au nombre à la ligne 0 et à la colonne a de la matrice deux dimensions associée à Xtime.

## 2.6 AddRoundKey :

**Principe algorithmique :**

Chaque case d'un bloc de données (i,j) sera modifiée en appliquant un XOR entre la donnée de la case concernée et la case d'indice (i,j) de la clé. Le schéma suivant illustre cette transformation :




**Spécification :**

La procédure `AddRoundKey()` prend en paramètre le tableau contenant les blocs de données, la clé de chiffrement et le nombre de bloc de données.

## 2.7 Key Schedule

Key Schedule est une procédure qui est appelée après chaque Round, elle calcule des sous-clés. Elle est constituée de deux éléments : la key expansion et la round key selection. La key expansion spécifie comment ExpandedKey est dérivée de la clé de chiffrement. Le nombre totale de bits dans ExpandedKey est égale à la longueur du bloc multipliée par le le nombre de rounds plus un, car la clé requière une round key pour l'addition de la clé initiale, et une pour chaque rounds. Il faut noter que la ExpandedKey est toujours dérivée de la clé de chiffrement.

### 2.7.1 Critère de conception

La clé d'expansion a été choisi celon les critère suivants :

1. Efficacité.
   - Mémoire utilisée : Il est possible d'exécuter la key schedule en utilisant une faible quantité de mémoire.
   - Performance : Elle doit avoir une grande performance sur un grand nombre de processeurs.
2. Elimination de la symetrie : Elle doit permettre de supprimer les symétries.
3. Diffusion : Elle doit avoir une diffusion efficace des différences de clé de chiffrements dans la clé d'expansion.
4. Non-linérité : Elle doit présenter assez de non-linéarité pour prohiber la totalité des différences de détermination dans la clé d'expansion d'après les différences de la clé de chiffrement.

### 2.7.2 Explication de sa création

Suivant la clé on détermine :
- Nr : Nombre de rounds.
- Nb : Nombre de lignes, fixé à 4 pour AES.
- Nk : Nombre de colonnes de la clé.

| Clé | 128 | 192 | 256 |
|---|---|---|---|
| Nr | 10 | 12 | 14 |
| Nk | 4 | 6 | 8 |

### 2.7.3 RotWord

On prend à chaque fois la $Nk_{ieme}$ colonne de la clé. Puis l'élément de la première ligne est descendu à la dernière ligne et tous les autres éléments remontent d'une ligne.

### 2.7.4 SubBytes

Avec cette colonne obtenue, on applique une transformation avec la S-Box. On remplace la case[i,j], qui contient un nombre hexadécimal codé sur deux chiffres, soit XY, par le nombre à la ligne X et à la colonne Y de la S-Box.

### 2.7.5 Les Round Constantes : RCon

On prend la colonne i-4, la colonne obtenue précedement et la colonne j de la RCon. On incrémentera alors j.
On fait alors un XOR avec ces 3 colonnes.
Puis on met cette colonne à la place i.

### 2.7.6 XOR

Pour les colonnes dont leurs indices n'est pas divisible par Nk, on fait un XOR avec i-4 et i-1, i étant l'indice de la colonne courante. Ce qui nous donne la colonne i.
On effectue ces transformations Nr fois.

# 3 Le déchiffrement (Dimitri, Vincent et Julien)

Pour le déchiffrement nous avons besoin des fonctions inverses de celles du chiffrement. Nous avons donc codé les fonctions :

- InvSubBytes
- InvShiftRows
- InvMixColumns
- InvKeyExpansion

## 3.1 Présentation de l'algorithme de déchiffrement

Elle importe un fichier texte contenant le texte chiffré. On fait donc appel à la fonction de coupage en blocs qui crée une matrice 3 dimensions.
Cette fonction nécessite la clé de chiffrement, qui a servi au chiffrement. Une fonction permet de convertir cette clé qui est une chaine en une matrice 2 dimensions.
La fonction de dechiffrement est codée suivant ce schéma : Tour est le nombre de tour de boucle, il dépend de la longueur de la clé :

- tour = 10, pour une clé 128 bits
- tour = 12, pour une clé 192 bits
- tour = 14, pour une clé 256 bits

1. Création de l'expansion de la clé.
2. AddRoundKey
3. InvSubBytes
4. InvShiftRows
5. Boucle de 1 à Tour
   - AddRoundKey
   - InvMixColumns
   - InvSubBytes
   - InvShiftRows
6. AddRoundKey

Il faut alors exporter le tableau 3 dimensions contenant les données déchiffrées, dans un fichier texte.

### 3.2 AddRoundKey :

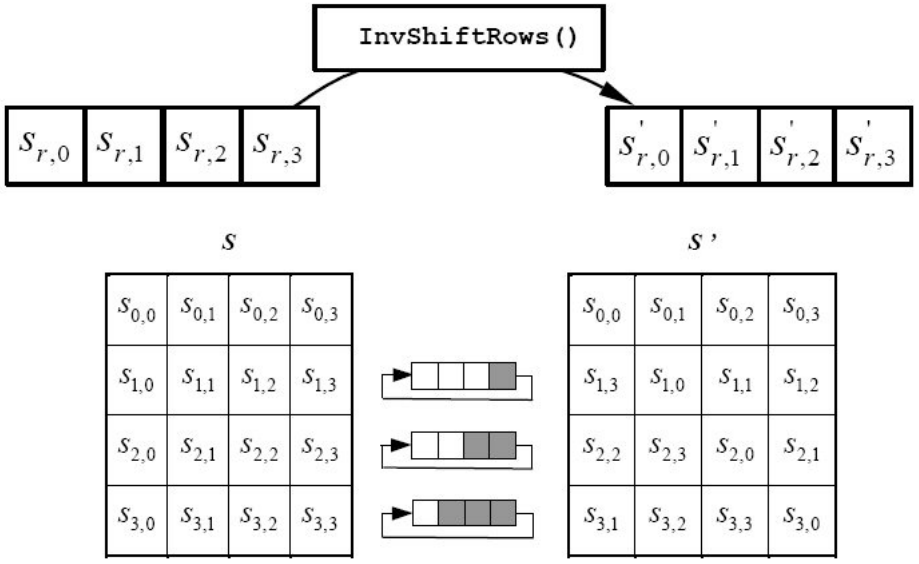
La fonction `AddRoundKey()` pour le déchiffrement est la même que celle du chiffrement.

### 3.3 InvSubBytes :

`InvSubBytes()`, elle, représente la fonction inverse de `SubBytes()`. Cette fonction inverse reprend le même principe que `SubBytes()` mais cette fois-ci en utilisant une SBox inverse, c'est-à-dire une SBox contenant des valeurs permettant de retrouver le texte d'origine.

### 3.4 InvShiftRows :

Comme son nom l'indique, la fonction `InvShiftRows()` est la fonction inverse de `ShiftRows()`. Elle reprend les mêmes transformations de rotation que cette derniere, mais dans le sens contraire afin de retrouver l'état original du state.



### 3.5 Travail effectué pendant cette soutenance

La partie sur le chiffrement étant finie, il a fallu finir la partie sur le déchiffrement. Nous avons du modifier la fonction KeySchedule, pour qu'elle ne calcule qu'une fois la clé, alors qu'elle la recalculait à chaque tour.

Nous avons également modifié la fonction de coupage des données à chiffrer et à déchiffrer, premièrement recoder la fonction calculant la longueur du fichier parceque l'ancienne était inadaptée pour un fichier chiffré constitué que de caractères spéciaux.
Un autre problème est apparue, il y avait des caractères spéciaux à la fin du fichier déchiffré, il a donc fallu modifier la fonction de coupage.

Une fonction de validation de la clé a été codé, elle vérifie la taille de la clé et vérifie si tous les caractères sont en hexadécimal. Si il y a plus de un espace entre les caractères, la fonction les supprime.

# 4 Stéganographie (Vincent et Sylvain)

## 4.1 Stéganographie dans un fichier BMP (Vincent)

La stéganographie est l'art de dissimuler des informations. En informatique et plus précisément pour notre projet nous dissimulons des données dans un fichier image.
Nous utilisons des fichiers BMP, qui sont les fichiers images permettant de stocker le plus de données.

### 4.1.1 Le principe

Les données à dissimuler sont stocker dans un fichier texte sous forme de caractères.

Un fichier BMP est constitué d'une entête (header) contenant differentes informations sur l'image.

| Nom du champs | Longueur en octets | Signification |
|---|---|---|
| Identifier | 2 | Contient toujours l'octet B suivit de l'octet M |
| Filesize | 4 | Taille totale du fichier en octets |
| Reserved | 4 | Champs réservé, doit être égale à 0 |
| DataOffset | 4 | Octets séparant le début du fichier des données de l'image |
| HeaderSize | 4 | Taille en octets de l'header |
| Width | 4 | Largeur de l'image en pixels |
| Height | 4 | Hauteur de l'image en pixels |
| Planes | 2 | Nombre de plans |
| BitsPerPixels | 2 | Nombre de bits nécessaires pour représenter un pixel |
| Compression | 4 | Type de compression |
| BitmapDataSize | 4 | Taille en octets des données de l'image |
| HResolution | 4 | Résolution horizontale de l'image en pixels par mètre |
| VResolution | 4 | Résolution verticale de l'image en pixels par mètre |
| Colors | 4 | Nombre de couleurs dans l'image |
| ImportantColors | 4 | Nombre de couleurs importantes |

Dans un BMP 24 bits, un pixel est codé sur 24 bits, soit 3 octets chacun représentant une composante couleur : ROUGE, VERT, BLEU. Nous placerons donc un caractère des données à dissimuler dans quatre pixels, donc tous les trois octets. Ou si la taille du fichier texte est trop importante par rapport à l'image, dans tous les octets de l'image.

On place alors 2 pixels du caractère dans les deux bits de poids faible d'un octet de l'image.

Ainsi dans une image de 1600 x 1200 pixels, on peut dissimuler :

- Au minimum : 1600*1200/4 = 480000 charactères, donc un fichier de 480 Ko.
- Au maximum : 1600*1200*3/4 = 1440000 charactères, donc un fichier de 1,44 Mo.

Et ceci sans que la taille de l'image ne soit modifiée.

## 4.2 Stéganographie dans un fichier WAV (Vincent et Sylvain)

Nous avons ajouté une option au logiciel, permettant de dissimuler un fichier texte dans un fichier son Wave.

### 4.2.1 Le principe

Les données à dissimuler sont stocker dans un fichier texte sous forme de caractères.

Un fichier Wave est constitué d'une entête (header).

| Octets | Signification |
|---|---|
| 1 à 4 | Caractères 'RIFF' [52h 49h 46h 46h] identifiant le format |
| 5 à 8 | Longueur du groupe de données au format WAV |
| 9 à 16 | Caractères 'WAVEfmt ' identifiant le format WAV |
| 17 à 20 | Nombre d'octets utilisés après pour définir le format |
| 21 à 22 | Numéro de format du fichier |
| 23 à 24 | Nombre de canaux |
| 25 à 28 | Fréquence d'échantillonnage (en Hz), c'est à dire le nombre d'échantillons pas seconde |
| 29 à 32 | Nombres d'octets par seconde |
| 33 à 34 | Produit du nombre de canaux par le nombre d'octets par échantillon |
| 35 à 38 | Bits par échantillon (valeurs possibles : 8, 12 ou 16) |
| 39 à 50 | Le nombre d'échantillons |
| 51 à 54 | 'data' : annonce l'arrivée des données |
| 55 à 58 | Taille des données |

Nous avons donc choisi de faire de la stéganographie dans des fichiers au format Wav, car c'est un format non compressé et le fait de le modifier légèrement ne change pas son aspect total et les modifications sont au final invisibles, ou plutôt inaudibles. Les données du son, les échantillons (8, 12 ou 16 octets de long), sont alignés les uns après les autres dans l'axe temporel (dans l'ordre où ils arrivent dans le temps).
Nous allons donc dissimuler les données dans chaque sample.
Pour réaliser cela nous placerons un caractère (1 octet) des données à dissimuler dans quatre échantillons. On ne modifiera que les deux bits de poids faibles. Il faut faire attention a ne pas modifier trops de bits sinon des parasites sonores sont perceptibles. Tout d'abord, nous devons connaître la structure du header, afin de commencer à modifier le fichier après celui-ci. Ensuite, on calcule la taille du fichier à cacher, et on vérifie que celle-ci soit suffisament petite. Nous allons ensuite cacher cette taille juste avant de commencer à cacher le fichier en lui même. Cela va nous permettre, lors de la décompression, de savoir le nombre d'octets à extraire pour reconstituer le fichier caché. Puis on cache le fichier lui même.

# 5 Algorithme Deflate (Dimitri et Julien)

Cet algorithme est la base de Gzip. Il utilise 2 méthodes de compression (sans perte de données) les plus connues : LZ77 et Huffman. Nous nous sommes inspirés de Deflate afin de nous rapprocher le plus du principe. Pour cette soutenance nous avons implémenté la décompression.

Dans un premier temps, LZ77 se charge de compresser le fichier, vient ensuite Huffman. Pour la décompression on utilise d'abord l'algorithme de Huffman puis LZ77.

## 5.1 Compression Lempel-Ziv (LZ77) (Dimitri)

Avant de rentrer dans les details de l'algorithme LZ77, il faut savoir qu'il existe 2 méthodes de compression sans perte de données :

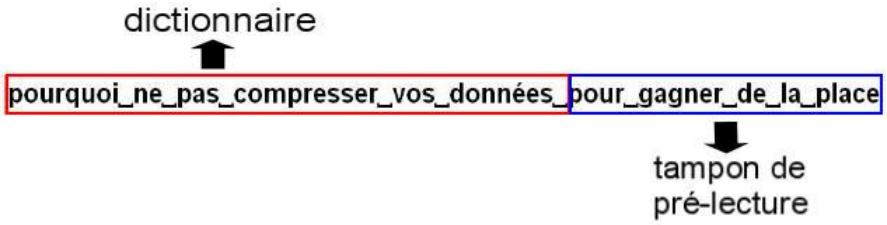
- les méthodes statiques (Huffman)
- les algorithmes de type dictionnaire (Lempel-Ziv)

Dans cette partie, on va s'intéresser aux algorithmes Lempel-Ziv, basés sur un article pubxlié en 1977 par les auteurs Jacob Ziv et Abraham Lempel. Ces algorithmes, dits algorithmes de dictionnaire, vont remplacer les chaines de caractères les plus présentes par un couple de $n$ valeurs (n variant selon l'algorithme). A partir de là, plusieurs variantes de l'algorithme Lempel-Ziv (LZ) ont été mis en place (LZ77, LZSS, LZ-78 Lempel-Ziv Welch, LZW, etc ...).

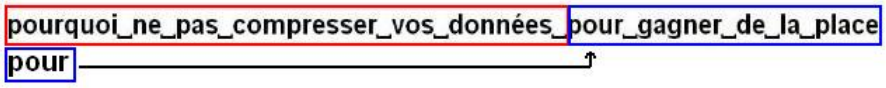
Afin de se rapprocher au plus de l'algorithme Deflate (compression Gzip), nous nous intéressons à l'algorithme LZ77. Cette méthode utilise deux fenêtres coulissantes : une première fenêtre qui servira de dictionnaire, et la seconde (appelée tampon de pré-lecture) servira à "repérer" les chaines identiques dans le dictionnaire. Si la chaine (ou une sous-chaine) du tampon de pré-lecture se retrouve dans le dictionnaire, alors elle est remplacée par un couple de 2 valeurs, la première correspondant à la position de la chaine du tampon de pré-lecture dans le dictionnaire, et la dernière à la taille de cette chaine (le vrai algorithme LZ77 utilise un couple de 3 valeurs. La $3^{eme}$ contenant le caractère suivant le tampon, cette dernière nous étant pas utile ne sera donc pas implémenté dans notre algorithme. Cette petite modification nous rapproche de LZSS).

Le principe est le suivant :

On dipose de 2 fenêtres. La première servira de dictionnaire au tampon de pré-lecture (deuxième fenêtre) comme le montre le schéma suivant :




Du tampon de pré-lecture, on recherche si la chaine (ou une sous-chaine) est présente dans le dictionnaire :




Si la (sous-)chaine exite, elle est alors remplacée par le couple de valeurs. On obtient donc :


Une fois terminée, on recommence la procédure $x$ caractères plus loin ($x$ correspond à la taille de la chaine la plus longue trouvée dans le tampon de pré-lecture. Si aucun chaine n'a été trouvée, x correspond alors à la taille du tampon de pré-lecture).

### 5.1.1 Algorithme LZ77

*Principe :*
Voir paragraphe précédant.

```
Algorithme Procedure Compression_LZ77
    Parametres Locaux
        chaine fichier
    Variables
        T_VectChar  dico, buffer
        entier      j, cpt
        FILE        fichier1, fichier_sortie
        char        c
        booleen     compare

Debut

    Ouvrir(fichier1, lecture_seul)
    dico   <- init_dico()
    buffer <- init_buffer()
    Si non (Ouvrir(fichier_sortie, lecture_seul)) Alors
        /* On creer le fichier de sortie */
    Sinon
        /* Demande d'ecrasement du fichier existant */
    FinSi

    Tant (c!=EOF) Faire

        Pour j<-0 jusqu'a MAX_DICO Faire
            /* Remplir le dictionnaire avec le texte */
            /* et on le recopie dans le fichier de sortie */
        FinPour

        /* On sauvegarde la position du pointeur de fichier1 */

        Pour j<-0 jusqu'a MAX_BUFFER Faire
            /* Remplir le tampon de pre-lecture avec le texte */
        FinPour
        cpt <- MAX_BUFFER
        compare <- faux
```

```
        Tant que (cpt>1) et non (compare) Faire

            compare <- /* recherche d'une (sous-)chaine dans le tampon */
                       /* de pre lecture presente dans le dico */
            cpt <- cpt-1 /* le compteur sert a extraire une sous-chaine */

        FinTantQue

        Si compare Alors
            /* On remplace la (sous-)chaine par le couple de valeurs */
            /* en ecrivant dans le fichier de sortie */
            /* On deplace le curseur de n caractere, n etant la taille */
            /* de la (sous-)chaine presente dans le dico */
        Sinon
            /* On positionne le curseur apres le tampon de pre-lecture */
        FinSi

    FinTantQue

/* Fermeture et libération de la mémoire */

Fin Algorithme Procedure Compression_LZ77
```

## 5.2 Compression Huffman (Julien)

### 5.2.1 Présentation de Huffman

La méthode de compression Huffman consiste à diminuer au maximum le nombre de bits utilisés pour coder un fragment d'information. Prenons l'exemple d'un fichier de texte : Le fragment d'information sera un caractère ou une suite de caractères. Plus le fragment sera grand, plus les possibilités seront grandes et donc la mise en oeuvre complexe à exécuter.
L'algorithme de Huffman se base sur la fréquence d'apparition d'un fragment pour le coder : plus un fragment est fréquent, moins on utilisera de bits pour le coder.

### 5.2.2 Fonction Huffman

Pour pouvoir compresser puis décompresser l'information, on va donc devoir utiliser une table de fréquences : pour cela nous allons calculer cette table puis nous générerons une table de correspondance qui sera intégrer dans le fichier. La compression est meilleure puisqu'elle est adaptée au fichier à compresser, mais elle nécessite le calcul d'une table de fréquences.

Specification : la fonction Huffman prend en paramètres le nom du fichier texte à compresser et le nom du fichier texte cible, elle retourne un entier afin de savoir si la compression a été effectué. Cette fonction fait appelle à plusieurs autres fonctions et types définies.

Principe algorithmique : la fonction Huffman fait appelle à la fonction fréquence afin de générer la table de fréquence des caractères présents dans le fichier à compresser. La table de fréquece se présente sous la forme d'une liste chainée :

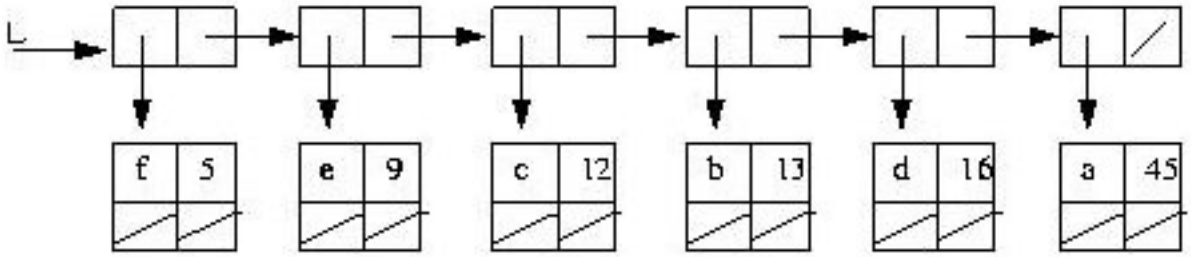



**Figure 2:** liste de construction de l'arbre binaire de Huffman, état initial

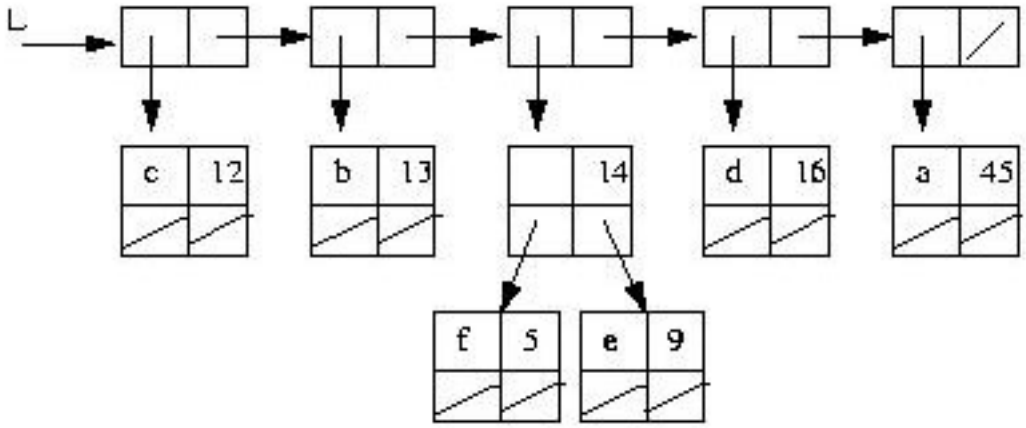
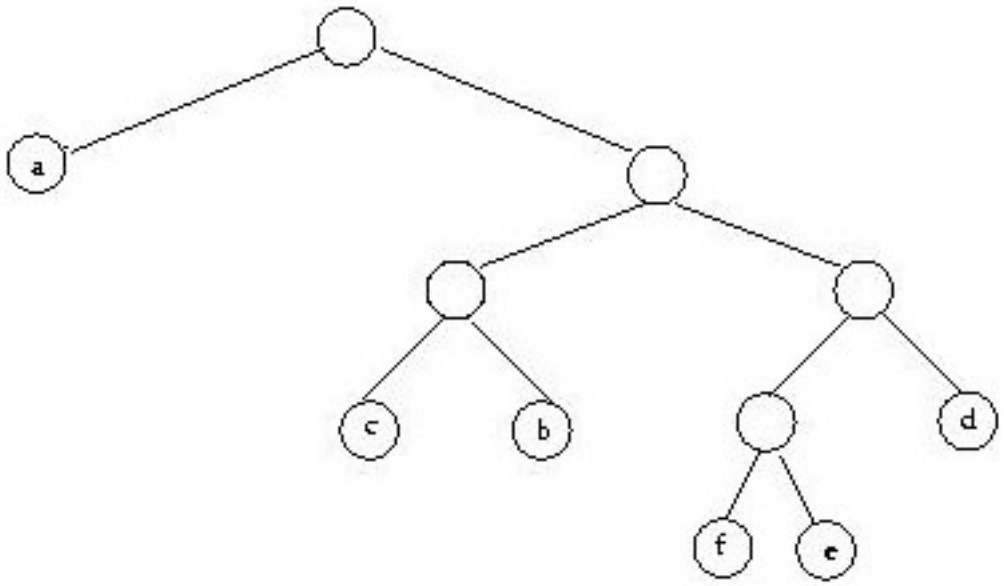



**Figure 3:** liste après la première fusion

Les élément de poids le plus faible sont en tëte de liste.

On génére ensuite l'arbre de Huffman à partir de la liste ordonnée, on prend les deux noeuds de poids les plus faibles et on les ajoute comme fils d'un nouveau noeud qui aura pour poids la somme des deux. Il suffit de réitérer cette étape jusqu'à ne plus avoir qu'un seul noeud. Après cela, descendre vers la gauche équivaut à un 0, et descendre vers la droite à un 1.



L'arbre de Huffman permet de créer la table de correspondance, pour cela on vas effectuer un parcours profondeur de l'arbre puis lorsqu'on est sur une feuille le valeur du chemin parcouru sera associé à la clé de la feuille qui représente un caractère. L'ensemble de ces valeurs son stockées dans un tableau. Après avoir parcouru l'arbre, le tableau que l'on obtient n'est autre que la table de correspondance.

| caractère | a | b | c | d | e | f |
|---|---|---|---|---|---|---|
| code Huffman | 0 | 101 | 100 | 111 | 1101 | 1100 |
| autre code | 000 | 001 | 010 | 011 | 100 | 101 |
| fréquence | 45 | 13 | 12 | 16 | 9 | 5 |

Il ne nous reste plus qu'à écrire la table de correspondance dans le fichier cible puis d'écrire le nombre de caractères présents dans le fichier à compresser, et enfin saisir chaque caractère du fichier regarder sa valeur dans la table de correspondance puis écrire sa valeur en binaire dans une variable de type char. On écrit la valeur de chaque caractère les uns à la suite des autres dans la variable, si on atteint le bit de poids fort (le huitième bit) alors on écrit, en mode binaire, dans le flux la valeur de cette variable. Si il n'y a plus de caractères à saisir dans le fichier et que l'on a pas atteint le bit de poids faible de la variable alors on affecte la valeur 0 aux bits restant.

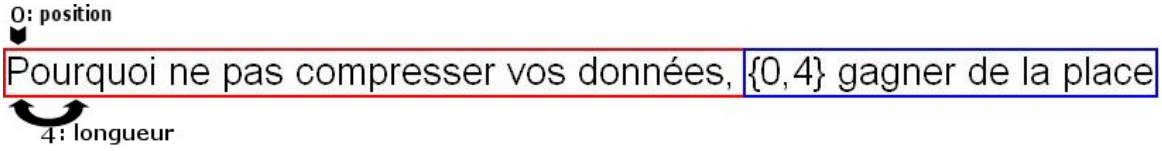
## 5.3 Decompression LZ77 (Dimitri)

Avant d'expliquer la méthode de décompression LZ77, voici un petit rappel du principe de compression :

On dispose d'une fenêtre appelée dictionnaire et une fenêtre appelée tampon de pré-lecture. Soit la phrase suivante : **"Pourquoi ne pas compresser vos données, pour gagner de la place ?"**. La première fenêtre (dictionnaire) contiendra le bout de phrase **"Pourquoi ne pas compresser vos données, "**, et la seconde (tampon de pré-lecture) contiendra le reste de la phrase, à savoir **"pour gagner de la place ?"**. Le principe est le suivant : on recherche une chaine, ou sous-chaine équivalente. Lorsqu'une équivalence est trouvée, elle est remplacée par un couple d'entier représentant respectivement la place de la chaine équivalente dans le dictionnaire, et la taille de cette chaine.

Maintenant, la phase de décompression. Elle diffère de très peu par rapport la compression. La méthode se devine assez facilement : lorsque l'on rencontre un couple d'entier, on teste si ce dernier est valide, si il l'est, alors on utilise les entiers afin de retrouver les chaines équivalentes dans le dictionnaire.


Dans l'exemple ci-dessus, on va se placer en position 0 dans le dictionnaire, soit à la premiere lettre, puis on va recopier la sous-chaine du dictionnaire de longueur 4 (jusqu'à la lettre 'r').



L'algorithme de décompression est le suivant :

```
Algorithme Procedure Decompression

    Parametres locaux

Debut

    Ouvrir(fichier1, lecture_seul)
    dico   <- init_dico()
    buffer <- init_buffer()
    Si non (Ouvrir(fichier_sortie, lecture_seul)) Alors
        /* On creer le fichier de sortie */
    Sinon
        /* Demande d'ecrasement du fichier existant */
    FinSi

    Tant (c!=EOF) Faire

        Pour i<-0 jusqu'a MAX_DICO Faire
        /*On remplie le dictionnaire*/
        /*On le recopie dans le fichier de sortie*/
    FinPour

    /*On sauvergarde la position actuelle*/
    /*On recupere le caractere actuel du fichier1 dans une variable c*/
    Si c<>'{' Alors
        /*On recule le curseur de -1 position*/
        Pour i<-0 jusqu'a BUFFER-1 faire
            /*On reecrit le texte normalement dans le fichier de sortie*/
        FinPour
    Sinon
        Pour i<-0 jusqu'a BUFFER-1 Faire
            /*On remplie le buffer*/
        FinPour
        /*On recupere les coordonnees (position+longueur)*/
        /*On recupere la sous-chaine se trouvant dans le dictionnaire*/
        /*à l'aide des coordonnees*/
    FinSi

    FinTantQue

Fin Algorithme Decompression
```

## 5.4 Fonction DecHuffman

La décompression est une étape primordiale car elle doit permettre â un utilisateur de retrouver ses données sans qu'elles aient été altéré.
Pour pouvoir décompresser l'information on va devoir dans un premier temps générer l'arbre de huffman â partir de la table de correspondance.Puis on parcours l'arbre en fonction des bits des caractéres compressés.

Specification : la fonction DecHuffman prend en paramètres le nom du fichier texte à décompresser et le nom du fichier texte cible, elle retourne un entier afin de savoir si la compression a été effectué. Cette fonction fait appelle à plusieurs autres fonctions et types définies.

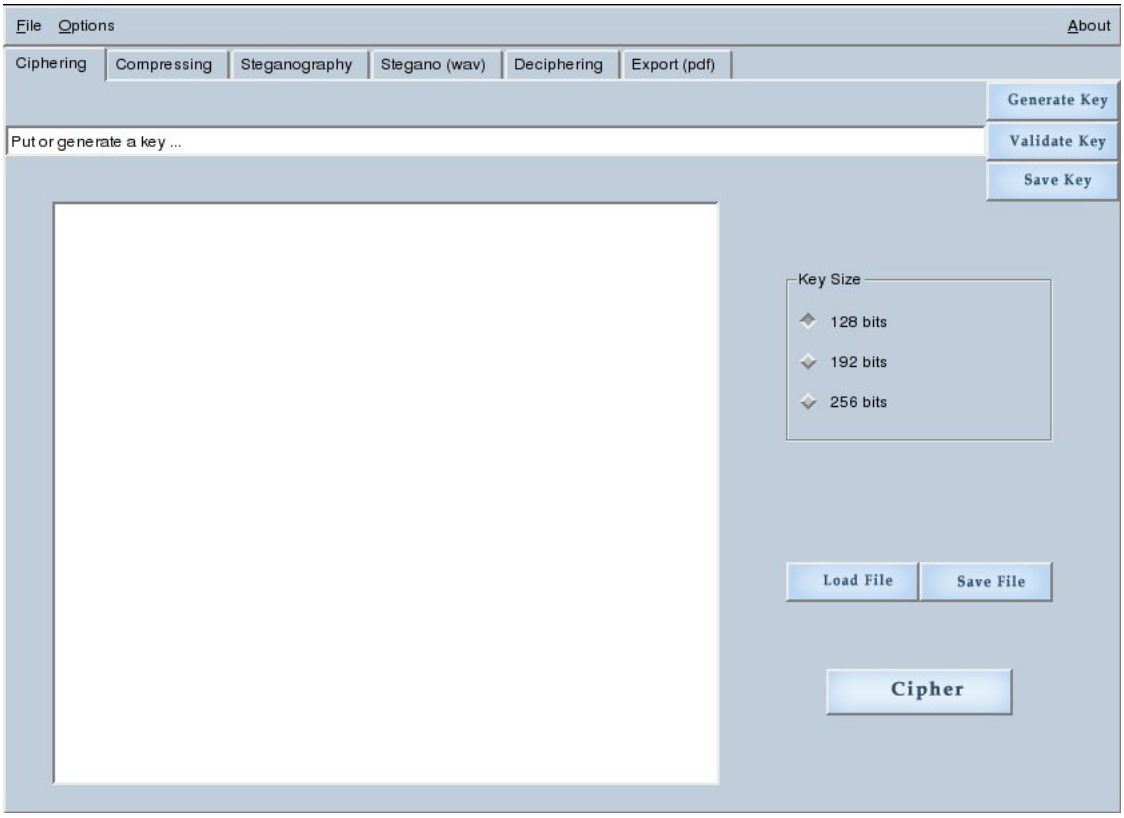
Principe algorithmique : la table de correspondance présente dans le fichier vas nous permettre de générer l'arbre.

Les caractéres de la colonne de gauche représentent les éléments des feuilles de l'arbre et les caractéres de la colonne de droite, l'ensemble de 0 et 1, représentent le chemin â parcourir dans l'arbre. On vas donc devoir saisir les caractéres de la table de correspondance et faire le trie afin de savoir lesquels correspondent â l'élément de l'arbre et ceux qui correspondent aux chemins.
Pour cela on utilise des compteurs d'espace et de saut à la ligne afin de détecter les éléments et les chemins.
Pour savoir si l'on a atteint la fin de la table il suffit de detecter plus de 3 sauts à la ligne consécutifs. Les chiffres ensuite représentent le nombre de caractéres dans le fichier cible. Les caractéres qui suivent sont les données compressées.

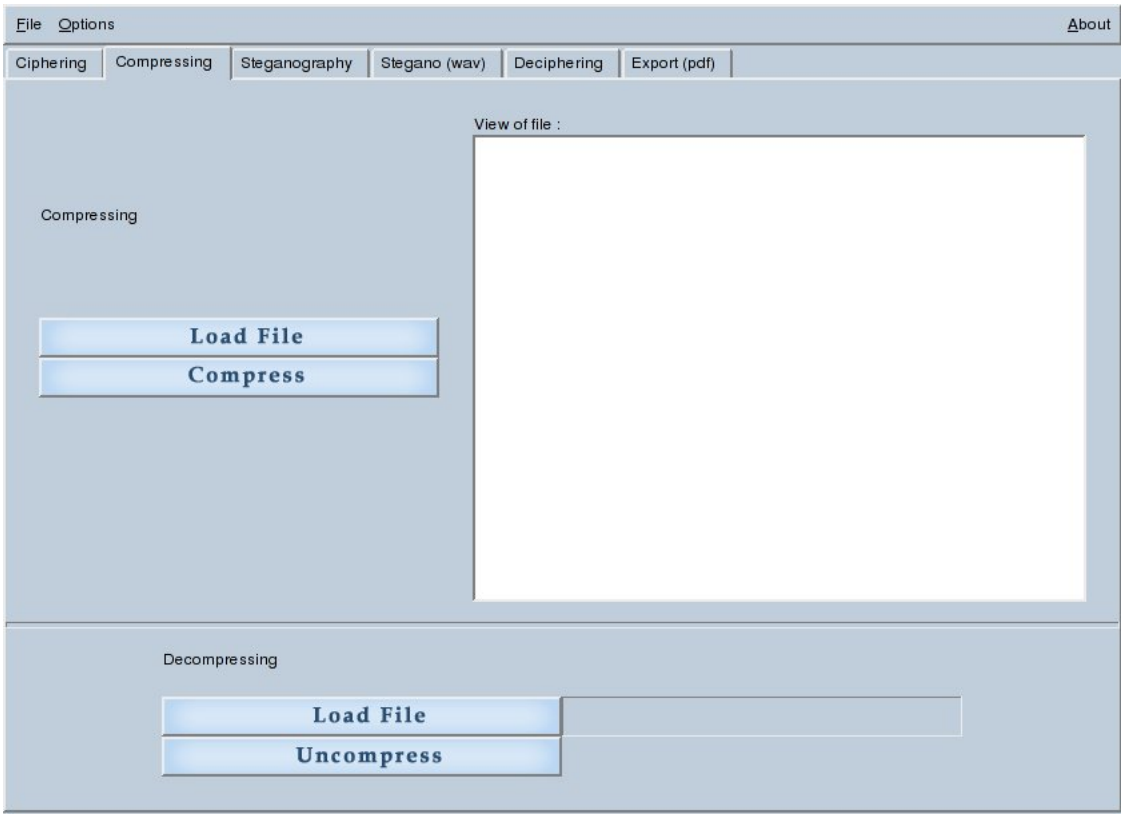
Le déchiffrement de ces caractères se réalise à l'aide de la fonction itoa_base qui prend en paramètre un unsigned char et le convertie en une chaine de caractéres en base 2. Cette chaine représente le chemin à parcourir dans l'arbre. On écrit le caractére du noeud courant dans le fichier cible et on décrémente le nombre de caractères quand on est sur une feuille, puis on réitère l'opération tant que le nombre de caractéres est positif.

# 6 L'interface (Sylvain)

L'année dernière, nous avions travaillé avec Delphi, qui permettait de réaliser très simplement des interfaces graphiques. Le placement des boutons ou autre se fesait d'un simple clic. Cette année, nous travaillons en C, il n'y a aucune interface graphique, la programmation se fait directement dans un éditeur de texte.
Pour réaliser l'interface graphique de notre logiciel, nous allons donc nous servir de bibliothèques déjà existantes.
Comme nous l'on présenté les spé de l'année dernière, les deux principales biblioteques d'interfaces graphiques pour le C/C++ sont QT et GTK. QT se programme en C++ et GTK en C. Bien que nous apprenions le C en cours, nous avons choisi, après nous être un peu plus renseignés, de nous servir de QT, qui est donc vous l'aurez compris, une bibliothèque C++ permettant de réaliser des interfaces graphiques.

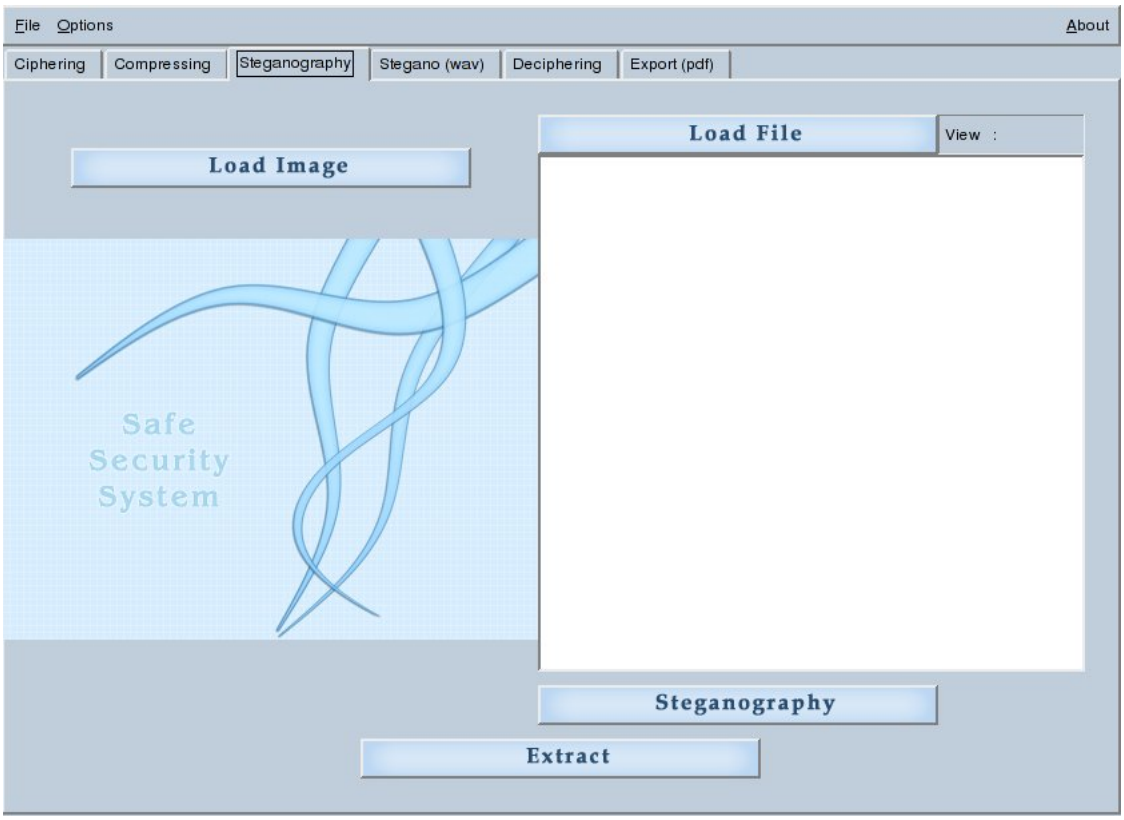


Pour la première soutenance j'avais simplement réalisé sur l'interface une barre de menu avec divers sous menus, comme ouvrir un fichier, sauvegarder, etc ... ces options n'étaient pas encore actives, elles étaient simplement affichées en prévision, afin de commencer à comprendre le fonctionnement de QT. Vincent avait codé un générateur de clé qui ppeut générer une clé de 128, 192 ou 256 bits. Afin que l'utilisateur puisse choisir la taille de la clé, j'avais placé dans l'interface une zone de 3 boutons pour permettre à l'utilisateur de sélectionner d'un simple clic la taille voulue.
J'avais ensuite placé des boutons : un appelant la fonction de Vincent et permettant ainsi de générer une clé, et un permettant de vérifier la clé (car celle ci est éditable) et de la sauvegarder pour l'utiliser. J'avais également mis provisoirement un bouton permettant d'executer la fonction de Julien, afin de pouvoir afficher le résultat et de montrer que celle-ci marche. La même chose pour quelques fonctions de Dimitri.
Pour l'affichage de la clé, j'avais tout d'abord affiché la clé générée directement dans une zone de texte non éditable mais comme il était prévu que l'utilisateur puisse modifier la clé obtenue, j'avais donc placé une zone de texte éditable dans laquelle la clé générée serait affichée. L'utilisateur peut alors la modifier librement. C'est pourquoi il y a ensuite un bouton de validation, afin de vérifier que la clé modifiée est toujours correcte.
Le plus long a été de trouver comment appeler les fonction de chacun qui sont en C dans l'interface qui est en C++. Il a fallu un peu de temps avant que ça marche, et pas mal de recherche. Mais finalement, lors de la première soutenance nous pouvions déjà présenter quelque chose de visuel et qui commencait à fonctionner même si cela ne ressemblait pas vraiment à l'aspect final que nous voulions donner au logiciel.

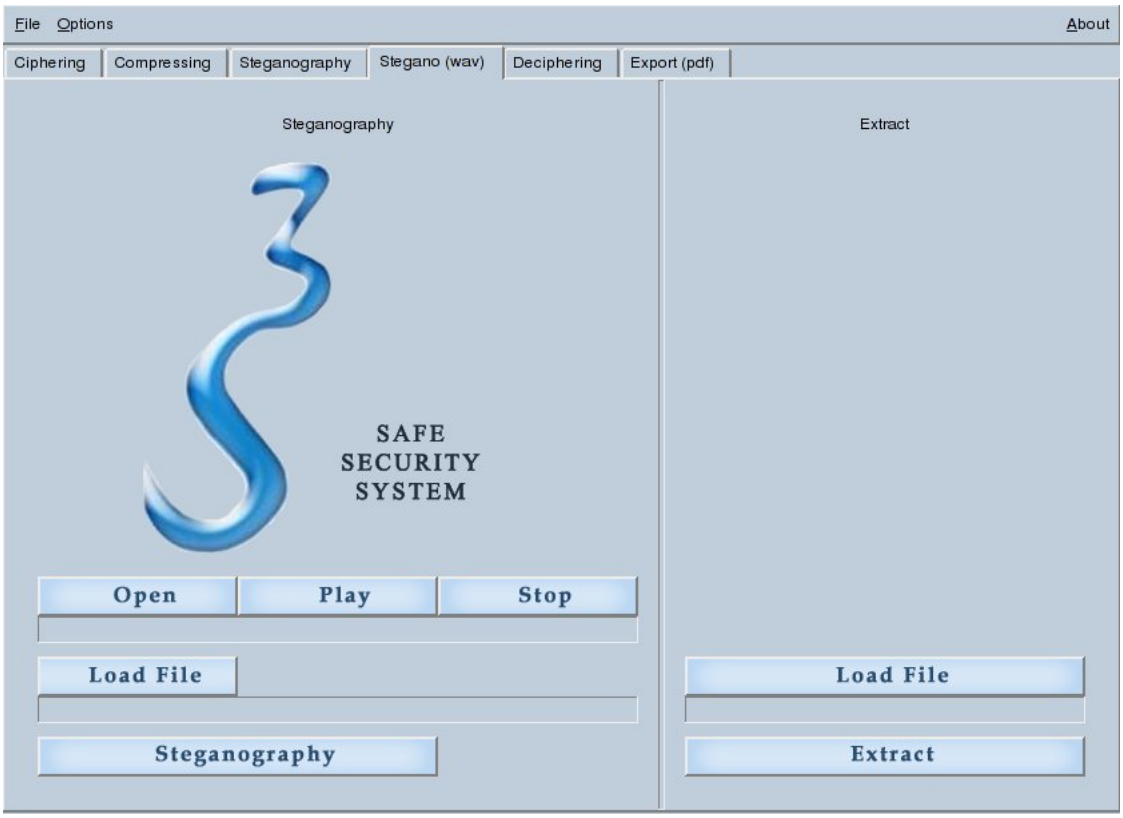



Pour la deuxième soutenance, étant déjà plus habitué à QT, j'avais donc réalisé une interface qui ressemblait fortement à l'interface finale que nous voulions donner à notre projet. Comme notre projet comprenait alors trois parties principales, (chiffrement, déchiffrement et stéganographie) il fallait que toutes ces fonctionnalitée soit accessibles facilement et rapidement. Nous avions donc choisi la technique des onglets. La fenêtre principale est donc composée d'une barre de menus et de trois onglets correspondant aux trois fonctionnalitées principales. Comme cela est particulier et ne pouvait pas s'intégrer directement à ce que j'avais déjà commencé pour la première soutenance, j'avais préféré tout recommencer à zéro, afin que celà soit bien structuré et plus facile à débugguer. Pour la partie chiffrement, l'utilisateur pouvait alors générer une clé, charger un fichier texte, et crypter ce fichier très simplement. Cette fois, les fonctions n'étaient plus séparées comme ce fut le cas à la première soutenance (puisque nous voulions les montrer en détails). Désormais, elles étaient regroupées pour ne faire qu'une seule fonction, qui allait chiffrer le texte préalablement chargé par l'utilisateur en se servant de la clé générée aléatoirement. Cela créait un nouveau fichier texte, qui contenait le texte chiffré. Il y avait donc sur l'onglet une zone d'aperçu de la clé générée, un bouton pour la générer, une zone d'aperçu du texte à chiffrer, un bouton pour charger celui-ci (via une inteface permettant d'explorer tout le contenu du disque dur), et évidement un bouton qui allait permettre de crypter le tout en lançant la boucle pricipale de chiffrement contenant toutes

les transformations nécessaires. Dans la partie stéganographie, il y avait la possibilité de charger une image (qui s'affiche en aperçu), de charger un fichier texte, puis grâce à un bouton, le texte va être intégré à l'image. Il y a bien sûr un bouton pour extraire le texte caché à l'intérieur d'une image. Celui-ci est placé dans un fichier texte créé pour l'occasion. La partie déchiffrement n'était alors pas encore implémentée. Nous avions donc une interface qui commençait à être fonctionnelle et utilisable ...

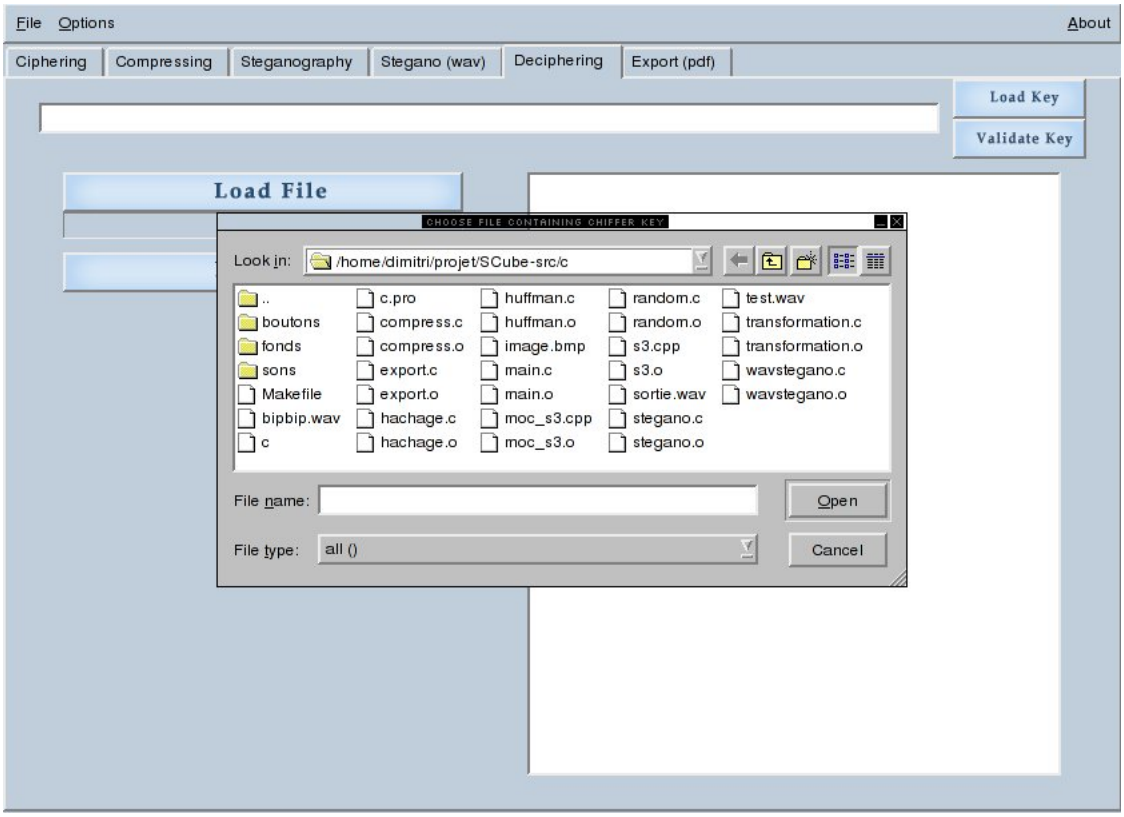


Pour la troisième soutenance, l'interface avait donc gardé l'aspect qu'elle avait à la deuxième. Les onglets stéganographie et stéganographie wave ont été ajoutés, des popups d'avertissement en cas de mauvaise manipulation ont été ajoutés, la plupart des fonctions avaient été implémentées et quelques autres ajouts avaient été fait, mais m'occupant également avec Vincent de la stéganographie dans un fichier audio, il n'y avait pas eu de modifications majeures dans l'interface.

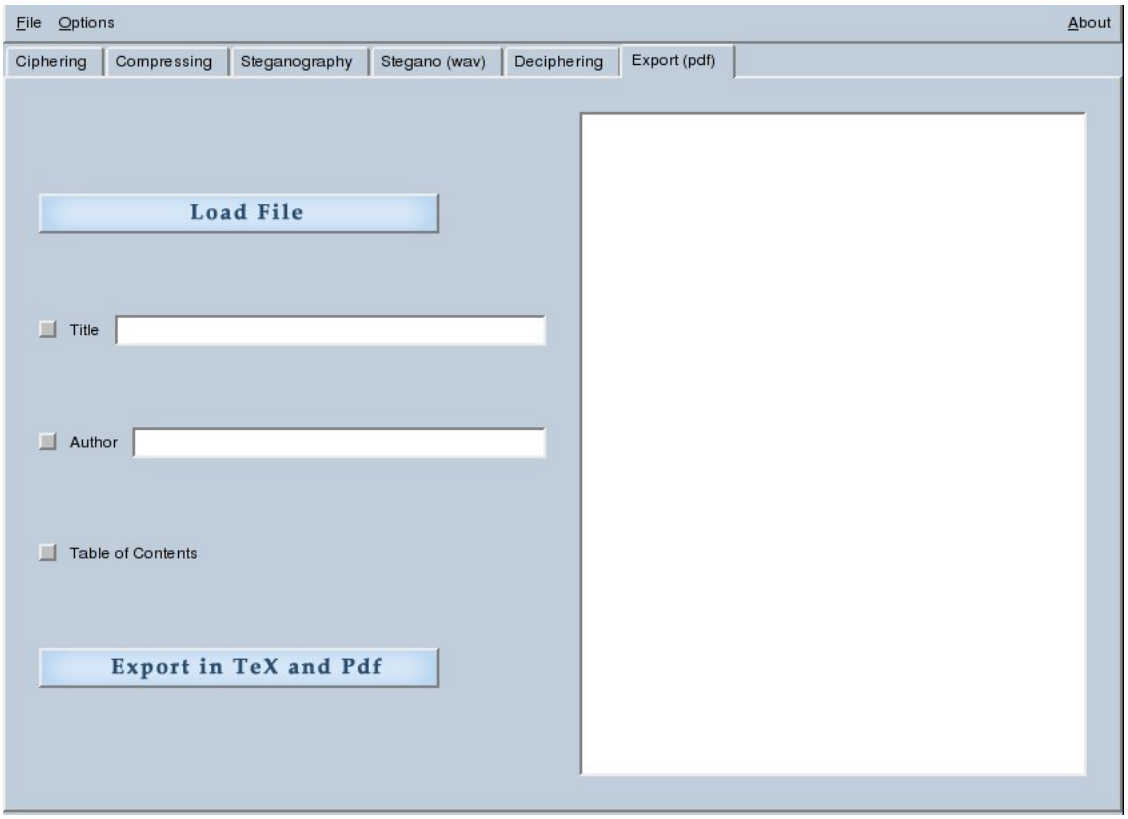



Enfin, pour la dernière soutenance, un gros travail a été réalisé sur l'interface, afin de rendre le logiciel extrèmement simple et convivial. J'ai rajouté la possibilité de redimensionner la fenêtre, ce qui s'est avéré assez long car je n'avais pas utilisé cette méthode dès le début, mais au final, l'interface est donc mieux organisée. Ensuite, nous avons passé pas mal de temps à changer l'apparence du logiciel : ajouts de fonds, remplacement de chaque boutons par des images afin d'avoir une interface qui ne ressemble pas à une interface grise vielliotte. Un long travail de débuggage a également été fait, car il y avait plusieurs endroits où les fonctions fesaient planter l'interface. J'ai également rajouté l'onglet afin de réaliser l'exportation d'un document en Latex (avec choix de l'auteur, du titre ...). L'onglet déchiffrement a enfin été remplis permettant d'acceuillir les fonctions tout justes terminées. Nous avons, pour encore plus de convivialité, rajouté des sons lors des clics sur les boutons ou encore lorsqu'une fonction a été lancée. Bref beaucoup de "petits détails" qui ont pris énormément de temps, en tout cas bien plus qu'on ne le

pensait. J'ai également voulu essayer de réaliser un portage windows. L'installation de QT, dont peu de versions gratuites sont disponibles pour cette plate-forme et contenant donc extrèmement peu de tutoriels (pour ne pas dire à peine un ou deux), m'a pris pas mal de temps. Après beaucoup de recherches j'ai néanmoins réussi à l'installer et même à compiler l'interface qui peut ensuite être affichée. Malheureusement, lors de l'intégration des fonctions, qui n'ont été codées que sous Linux, provoquent beacoup d'erreur, et nous n'avons pas le temps de les revoir toutes pour assurer la compatibilité windows. Nous avons donc abandonné l'idée.



Finallement, nous avons donc une interface très agréable visuellement, avec des boutons sympathiques, des onglets qui permettent une navigation facile et rapide entre les différentes fonctionnalitées du logiciel, des boutons intuitifs et rapides à utiliser, bref, l'interface est à la hauteur de notre logiciel et est conforme à ce que nous voulions au début.



# 7 Exportation en PDF/LaTeX(Julien et Dimitri)

L'exportation en fichier PDF/LaTeXpermet à l'utilisateur de convertir n'importe quel fichier texte (.txt) en fichier LaTeX(.tex) et PDF (.pdf).

*Spécification :*

La procédure `tex_export()` est une fonction à arguments variables :
- le premier argument est le nom du fichier texte à convertir
- le deuxieme argument sert à definir la présence d'une table de matière : si il est à 1, la table sera affichée, sinon elle ne le sera pas
- le troisième argument sert à definir la présence d'un titre : si il est à 1, le titre sera affiché, sinon il ne le sera pas
- le quatrième argument sert à definir la présence d'un auteur : si il est à 1, le nom de l'auteur saisie sera affiché, sinon il ne le sera pas
- le cinquième argument, optionel, est le titre saisie par l'utilisateur. Il sera affiché si le troisième parametre est à 1
- le sixième paramètre, optionel, est le nom de l'auteur. Il sera affiché si le quatrième argument est à 1

La procédure `tex_export()` convertie en fichier PDF/LaTeX.

*Principe :*

La procédure convertie les caractères spéciaux (tels que '{', '}', '#', '~', '&' ...) et laisse les autres caractères tels quel.
Elle convertie le caractère '\ n' par '\ \'. La procédure gère aussi la présence de section, sous-section etc...Elle repère un format spécifique qui est le suivant :

- 'x.' est le format d'une section (\ `section{}`)
- 'x.x.' est le format d'une sous-section (\ `subsection{}`)
- 'x.x.x.' est le format d'une sous-sous-section (\ `subsubsection{}`)

où x est un nombre.

Cette partie de la procédure compte le nombre de point présent dans un des formats : un point correspond à une section, deux points a une sous-section et 3 points ou plus à une sous-sous-section. Le format doit être suivi d'un espacement puis d'une chaine de caractères.

Si aucun des formats n'est reconu, alors ils sont traités comme des chaines de caractères.

A la fin de la procédure, l'appel à `system()` permettera de compiler directement le fichier LaTeX(.tex) afin d'avoir le fichier PDF final.

# 8 Le site web (Vincent)

Pour cette soutenance nous avons traduit la page d'accueil du site en créole, permettant une plus grande internationalisation du projet.
Nous avons également ajouter les derniers rapports de soutenances sur la page des données.

# 9 Joies et peines

## 9.1 Vincent

J'ai commencé par faire des recherches sur AES et sur la génération d'une clé aléatoire.
Les chambres de chacun des membres étant étroites cela rend les coding parties peu confortables, les derniers jours ont été trés chaud et il faisait une chaleur étouffante.

La principale difficulté de la stéganographie dans un fichier Wave, a été de comprendre sa structure, et de trouver un bon compromis entre le nombre de bits modifiés dans un échantillon et la qualité à l'écoute du fichier sonore.

La compréhension de Mixcolumns m'a posé des problèmes. L'explication fournis dans le livre officiel de Rijndael est confuse et en anglais.

Enfin le débogage de toutes les fonctions de chiffrement et de dechiffrement a posé de nombreux problèmes.

## 9.2 Dimitri

Les principales difficultées ont été rencontrées lors de la première et troisième soutenance. Etant rentré chez moi lors des vacances de Noel, il m'a été difficile d'avancer le projet avant les vacances car une fois arrivé chez moi en Guadeloupe, je savais deja que j'aurais eu du mal a travailler, surtout avec toutes ces jolies filles dénudées sur la plage sous un soleil frappant. De plus, mon vol a été retardé dut a une incompréhension entre la companie d'Air France et moi même, m'obligeant a rester quelques jours de plus sur mon île (quel malheur !).
A la troisieme soutenance, mon temps a été difficile a gérer entre les partiels et les autres matières. De plus, quelques bugs on été relevé de mon coté ainsi que du coté des autres membres de mon groupe qui nous obligaient a travailler d'avantage afin de finir a temps le projet $S^3$.

## 9.3 Sylvain

Pour la préparation de la première soutenance, c'est à dire en tout début de projet, j'ai été déçu de voir aussi peu de documentation française sur QT. La documentation QT est quant à elle très complète mais exclusivement en anglais. Il est donc peu aisé de naviguer dans l'explication des fonctions qui sont quand même assez complexes. Malgrès tout, après les lectures de quelques tutoriels, il est très plaisant de commencer à afficher quelque chose et de découvrir au fur et à mesure toutes les possibilités (qui sont énormes) de QT. Nous avons été, par contre, très déçus du café de Vincent et lui avons acheté des stages chez "El Gringo". Tout au long de l'année, le travail sur l'interface a provoqué des joies et des peines : On est content lorsqu'on arrive à intégrer une nouvelle fonction, à ajouter de nouvelle fonctions, mais il est parfois difficile de rester bloquer longtemps

fasse à un bug dont on ne sait comment se sortir. Malgrès tout, c'est dans l'ensemble très plaisant de voir avancer le logiciel petit à petit, au fur et à mesure des ajouts, surtout lorsque ça fonctionne. Cette année a donc été une nouvelle fois très enrichissante et reste au final un bon souvenir.

### 9.4 Julien

Certaine difficultés ont été rencontré notamant lors de l'implémentation de la fonction de coupage en blocs de données. Il a fallu se mettre d'accord avec les autres membres afin de déterminer le type de tableau que nous allions utiliser. Puis lors de la recherche d'informations sur AES, la difficulté intellectuelle de certaines transformations nous ont demandé une grande concentration et une grande patience afin de saisir toute l'ampleure de leurs signification.
L'implémentation et la correction des imperfections de la fonction de compressions ont été éprouvante mais la joie d'avoir réussi procure un sentiment de satisfaction. L'implémentation des fonctions de déchiffrement a posé de nombreux problémes algorithmiques en raison des différents lemmes compliqués qui nous sont présentés dans les algorithmes. Le projet $S^3$ a été une véritable réussite et mon seul reget est que l'on ait pas eu le temps d'inclure d'autres modules utiles.

# 10 Conclusion

Pour conclure, la réalisation de ce projet nous a été très bénéfique. En effet, nous avons pu mettre en pratique les algorithmes de hachages et de compression de donnés vu au cours de l'année. La complexité mathématique et algorithmique de la méthode de chiffrement Rijndael nous ont permis d'être confronté aux problémes que nous sommes amenés à rencontré dans notre vie d'ingénieur informaticien.

Un véritable travail de groupe a été réalisé au cours de l'année. Les réunions et le communication au sein de la Vincent-Team a permis au projet d'avancé dans les délais prévus. Les journées de code ensemble ont été très instructives et ont représenté pour nous de bons moments. $S^3$ est une véritable réussite technologique et humaine. Sa popularité, sa simplicité d'utilisation en feront le logiciel de chiffrement de référence.