S^3 - Safe Security System

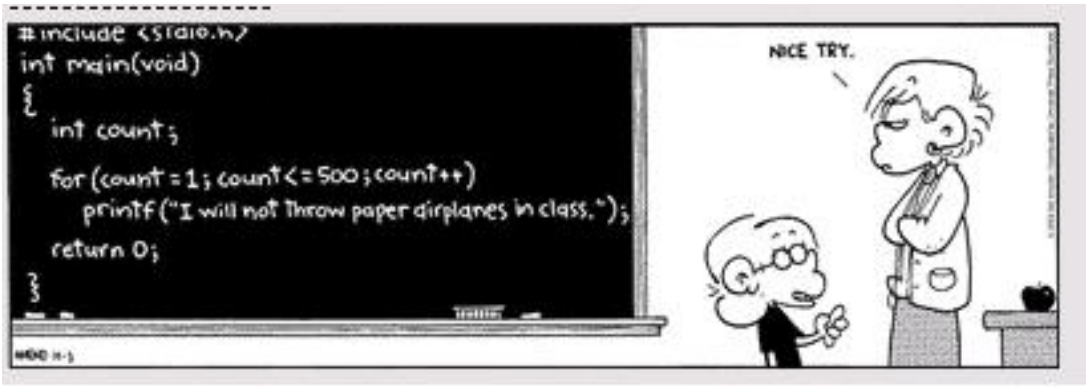
Bellone Sylvain - Mao Julien - Mérault Dimitri - Taront Vincent

Mai 2005

Table des matières

1 Introduction

Ce rapport de soutenance présente l'évolution de la création du logiciel Scube. Pour cette troisiéme soutenance les fontions de décompression, de stéganograhie dans un fichier son et d'exportation d'un fichier texte vers un fichier Latex ont été implémenté. Les différentes fonctions de déchiffrement ont aussi été implémenté.
Pour chaque domaine, les personnes concernées exposeront le travail réalisé et celui qui reste a faire ainsi que les différents problèmes rencontrés au cours de leurs travaux et leurs solutions.
Pour finir nous effectuerons un bilan de l'ensemble des travaux réalisés et de ce qu'il reste a faire.



2 Travail effectué

Cette section énumère le travail réalisé pour cette troisième soutenance dans chacun des domaines de travail.

2.1 Stéganographie.(Vincent et Sylvain)

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

2.1.1 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.

2.2 MixColumns(Vincent)

MixColumns a été recodée pour cette soutenance. 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.

2.3 Déchiffrement

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

- InvSubBytes
- InvShiftRows
- InvMixColumns
- InvKeyExpansion

2.3.1 La fonction 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.

2.4 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.

2.5 Décompression Huffman (Julien)

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.

Aprés avoir compresser le texte, le fichier compressé contient la table de correspondance puis le nombre de caractéres et enfin les caractéres compressés.

2.5.1 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.

```
2 10110111
d 10111
t 1100
a 1101
o 11100
- 111010
f 111011
i 1111

9369
´øN\230í\232&íz¬\236#fÅÌ^\ËF\206\IÞ¢u^H8Åäáß\227á¿ª\23
<^Å^PÊq¯ä^^¦^Eà|©\237]^DîL^K5^_$^G^RF½\204¢Î^V^ZwÓ»f\2
(øôé\214ùèùÜ#\202]è8Ñº÷8^O\ùô^V^Rwùåı>]»-^\-z¬\236#fÆ
J¯^N,^AL\223^O´^Tªçf£çp\216      w'khn^Mµ^Rv^_q#¦;\2237
yj§Z^U¸Ù¡J@h'\226-É^Qº^CG^C\211#S´|\224^V^Íw¨^ÝB^Nn}^E
^W\223  4°^M^N{æËA0,\223£7^M^\Ä\203\^D^K:^A*\237äs^G1¶
J±ßC-^DÅJús\^F\216^F¥{\212ÑN5ê²x^Í\233:^U^Z^ÐhÌÏ©à««f
'ÍÈøÊqò\216^F\202a¥};¶h\2314\206±f^Ï/\224p^@ ¼X\205\22
^G0]J(A\211A|Ð;¬^Í)À^TÉ^A ¥Í^\^Q)w'^M`  \206\225ôéÝô2
‡\206æ\200ŐàÛ[*^S\210Ő Q£ÅtÀÌ^P4^Eő-^^^S¦ul̃Ñ3E^Z<'L^\
^ÍH4GÔðUŐªDÍ^T6ÿß]cÈ^F^ÐDdxN\2309\202^G^[±^Ï ^ZA^Q\221
]ÍÃJúqÌ^\ÅLNOÎT\223¾\206Z5Éÿ    ª\234\2344y{    AWX/^U
```

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

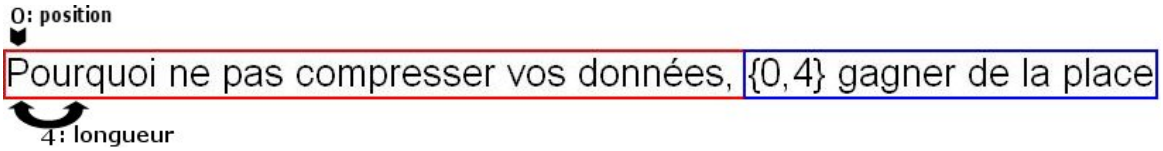
2.6 Decompression LZ77 (Dimitri)

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
```

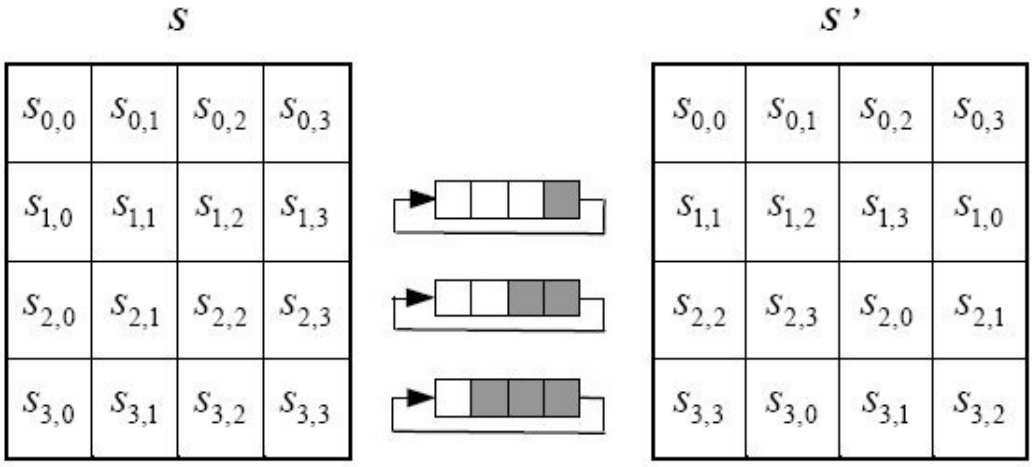
2.7 Exportation en format LaTeX(Dimitri)

Comme il l'a été dit dans le cachier des charges, il est également possible d'exporter ses fichiers texte en format LaTeX.

L'exportation de ce dernier est très simple. En effet, il s'agit de recopier un fichier dans un autre, en y ajoutant les entetes LaTeXcorrespondant. On commence par initialiser le fichier de sortie avec les entetes LaTeX, puis, par exemple, à chaque saut de ligne, on y rajoute un double backslash.

2.8 InvShiftRows (Dimitri)

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.



2.9 InvSubBytes (Dimitri)

`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.

2.10 InvKeyExpansion (Groupe)

Il existe plusieurs algorithmes InvKeyExpansion en fonction de la taille de la clé (128 bits ou plus) voici les algorithmes que nous avons trouvé dans le livre The Design of Rijndael :

```
InvKeyExpansion(byte K_i[4][N_k], byte W_i[4][N_b(N_r + 1)])
                                                    /* for N_k ≤ 6 */
{
for(j = 0; j < N_k; j++)
  for(i = 0; i < 4; i++) W_i[i][j] = K_i[i][j];
for(j = N_k; j < N_b(N_r + 1); j++)
  {
  if (j mod N_k == 0)
    {
    W_i[0][j] = W_i[0][j − N_k] ⊕ S[W_i[1][j − 1] ⊕ W_i[1][j − 2]] ⊕ RC[N_r + 1 − j/N_k];
    for(i = 1; i < 4; i++)
      W_i[i][j] = W_i[i][j − N_k] ⊕ S[W_i[i + 1 mod 4][j − 1] ⊕ W_i[i + 1 mod 4][j − 2]];
    }
   else
    {
    for(i = 0; i < 4; i++)
      W_i[i][j] = W_i[i][j − N_k] ⊕ W_i[i][j − N_k − 1];
    }
  }
  }
```

Voici l'algorithme pour un clé de 128 bits.

```
InvKeyExpansion(byte K_i[4][N_k], byte W_i[4][N_b(N_r + 1)])
                                                    /* for N_k > 6 */
{
for(j = 0; j < N_k; j++)
  for(i = 0; i < 4; i++) W_i[i][j] = K_i[i][j];
for(j = N_k; j < N_b(N_r + 1); j++)
  {
  if (j mod N_k == 0)
    {
    W_i[0][j] = W_i[0][j − N_k] ⊕ S[W_i[1][j − 1] ⊕ W_i[1][j − 2]] ⊕ RC[N_r + 1 − j/N_k];
    for(i = 1; i < 4; i++)
      W_i[i][j] = W_i[i][j − N_k] ⊕ S[W_i[i + 1 mod 4][j − 1] ⊕ W_i[i + 1 mod 4][j − 2]];
    }
  else if (j mod N_k == 4)
    {
    for(i = 0; i < 4; i++)
      W_i[i][j] = W_i[i][j − N_k] ⊕ S[W_i[i][j − N_k − 1]];
    }
  else
    {
    for(i = 0; i < 4; i++)
      W_i[i][j] = W_i[i][j − N_k] ⊕ W_i[i][j − N_k − 1];
    }
  }
  }
```

Voici l'algorithme pour une clé de plus de 128 bits.

2.11 InvMixColumn (Groupe)

Le déchiffrement a une structure similaire à celle du chiffrement, mais utilise la fonction InvMixColumn au lieu de MixColumn. Là où les coefficients de MixColumn sont limités à 01, 02 et 03, les coefficients de InvMixColumn sont 09, 0E, 0B et 0D. Ainsi chaque colonne est transformée en la multipliant par :

$$d(x) = (04x^2 + 05)c(x) \pmod{x^4 + 01}.$$

In matrix notation, this relation becomes:

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \times \begin{bmatrix} 05 & 00 & 04 & 00 \\ 00 & 05 & 00 & 04 \\ 04 & 00 & 05 & 00 \\ 00 & 04 & 00 & 05 \end{bmatrix}.$$

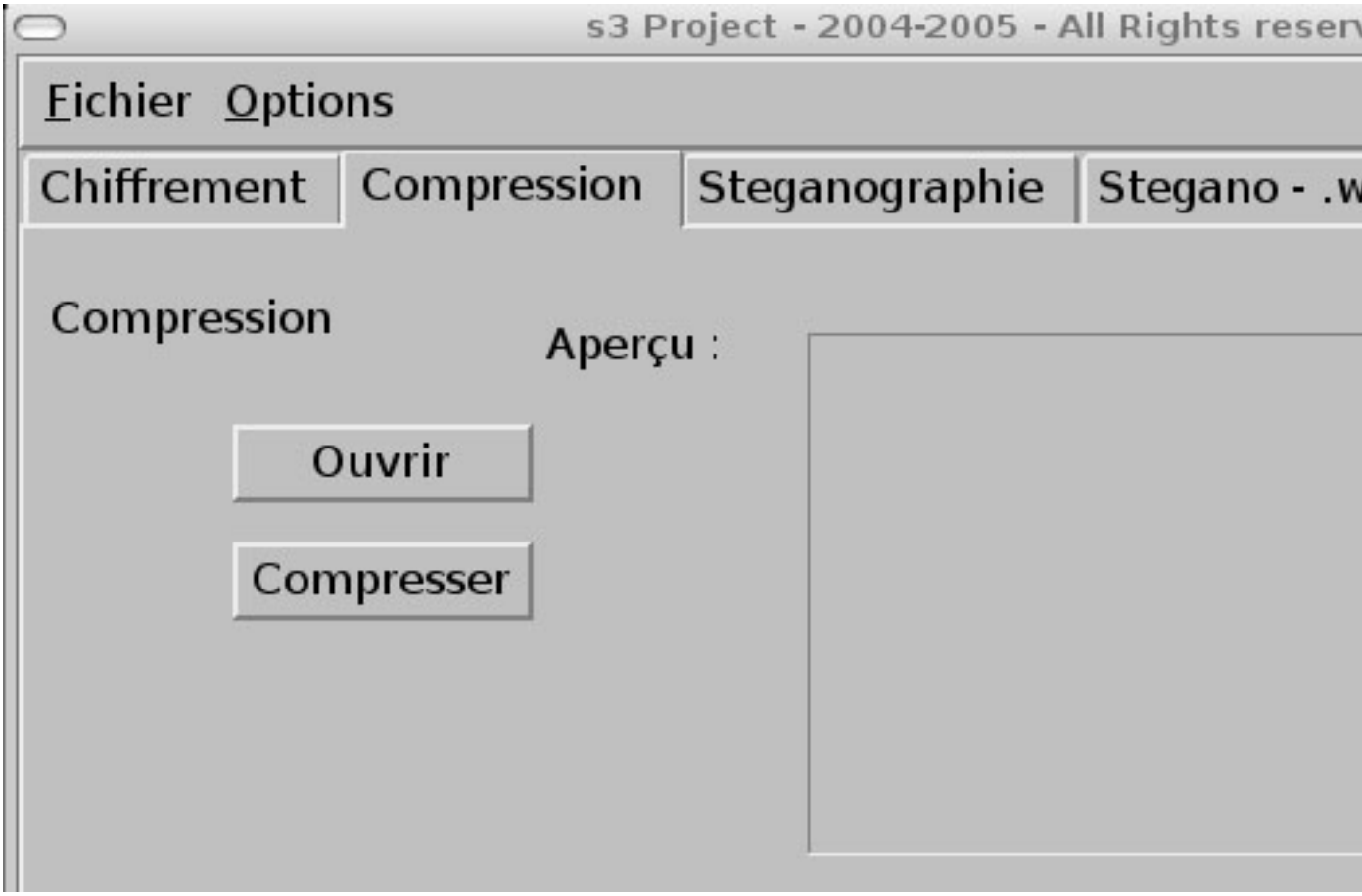
Ecrite comme une multiplication de matrices, InvMixColumn transforme les colonnes.
Voici l'algorithme que nous avons trouvé dans le livre The Design of Rijndael :

```
u = xtime(xtime(a[0] ⊕ a[2])); /* a is a column */
v = xtime(xtime(a[1] ⊕ a[3]));
a[0] = a[0] ⊕ u;
a[1] = a[1] ⊕ v;
a[2] = a[2] ⊕ u;
a[3] = a[3] ⊕ v;
```

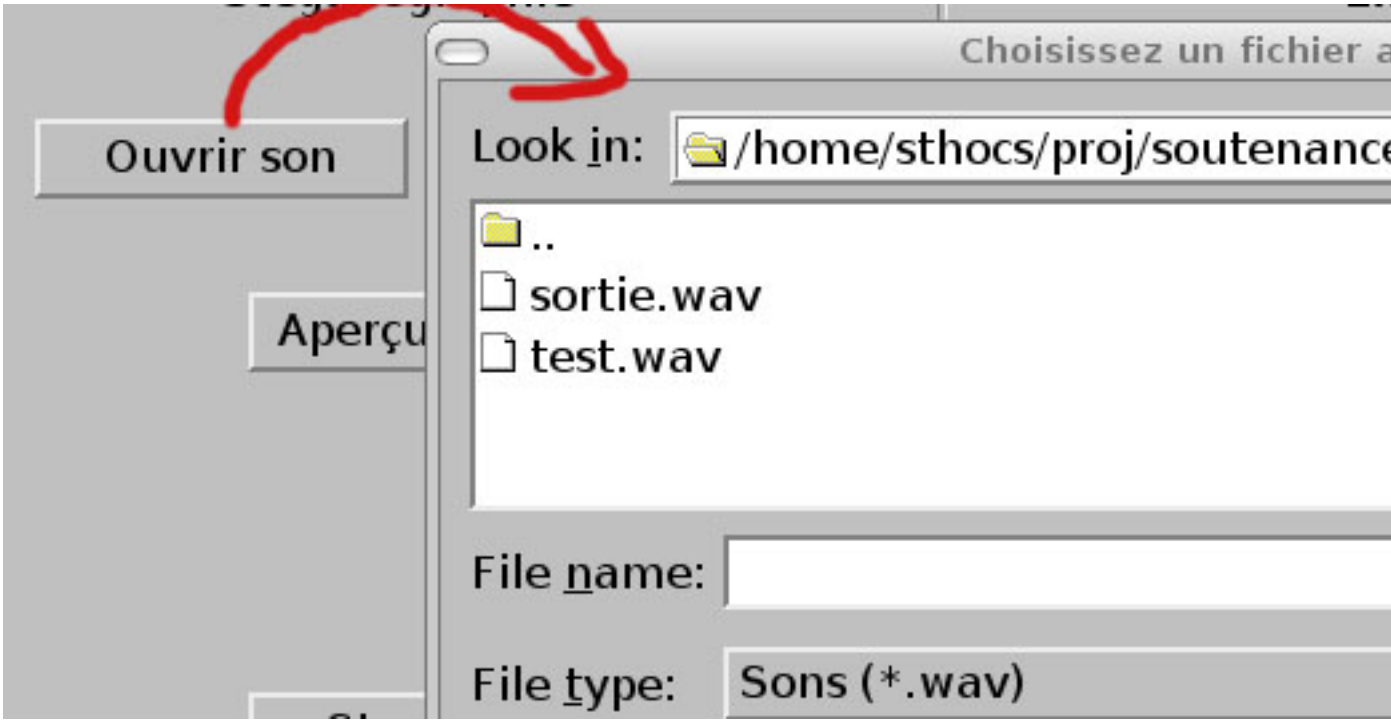
3 Interface (Sylvain)

Pour l'interface, la grande majorité des fonctions sont désormais codées, mais leur intégration s'avère assez longue. Le plus long n'est pas à chaque fois d'organiser les nouveaux boutons dans l'interface, car bien que ce soit fastidieux, j'ai maintenant l'habitude. Le problème est encore et toujours la compatibilité des fonctions écrites en C avec QT. En effet, pour les chaînes de caractères, QT utilise son propre type, qui est QString. C'est ce type là qui est renvoyé lorsque j'implémente une boîte de dialogue pour permettre à l'utilisateur de choisir le nom du fichier qu'il souhaite ouvrir, ou le nom qu'il souhaite donner à son fichier de sortie. Or en utilisant ce type dans les fonctions où les paramètres étaient des chaînes représentées traditionnellement en char*, cela donnait des erreurs. Il a donc fallu modifier les paramètres de certaines fonctions pour assurer la compatibilité. Le plus long à trouver à certainement été pour que les fonctions déclarées dans les headers soient reconnues dans QT. J'ai passé longtemps à ne pas comprendre ce qui n'allait pas, surtout que peu de forums parlaient de mon problème et il y avait encore moins de réponses. Heureusement je suis tombé par hasard sur une réponse qui suggérait une solution pour un problème similaire. Sans trop y croire j'ai essayé et ça a marché ... ouf. Bref, beaucoup de problèmes dans ce genre, ce qui fait qu'au final, la mise en place de l'interface m'a pris beaucoup de temps. D'ailleurs, certaines fonctions qui nous ont posées plus de problèmes que les autres n'ont pas encore pu être implémentées, faute de temps pour comprendre d'où venaient les problèmes... L'interface se compose donc de cinq onglets :

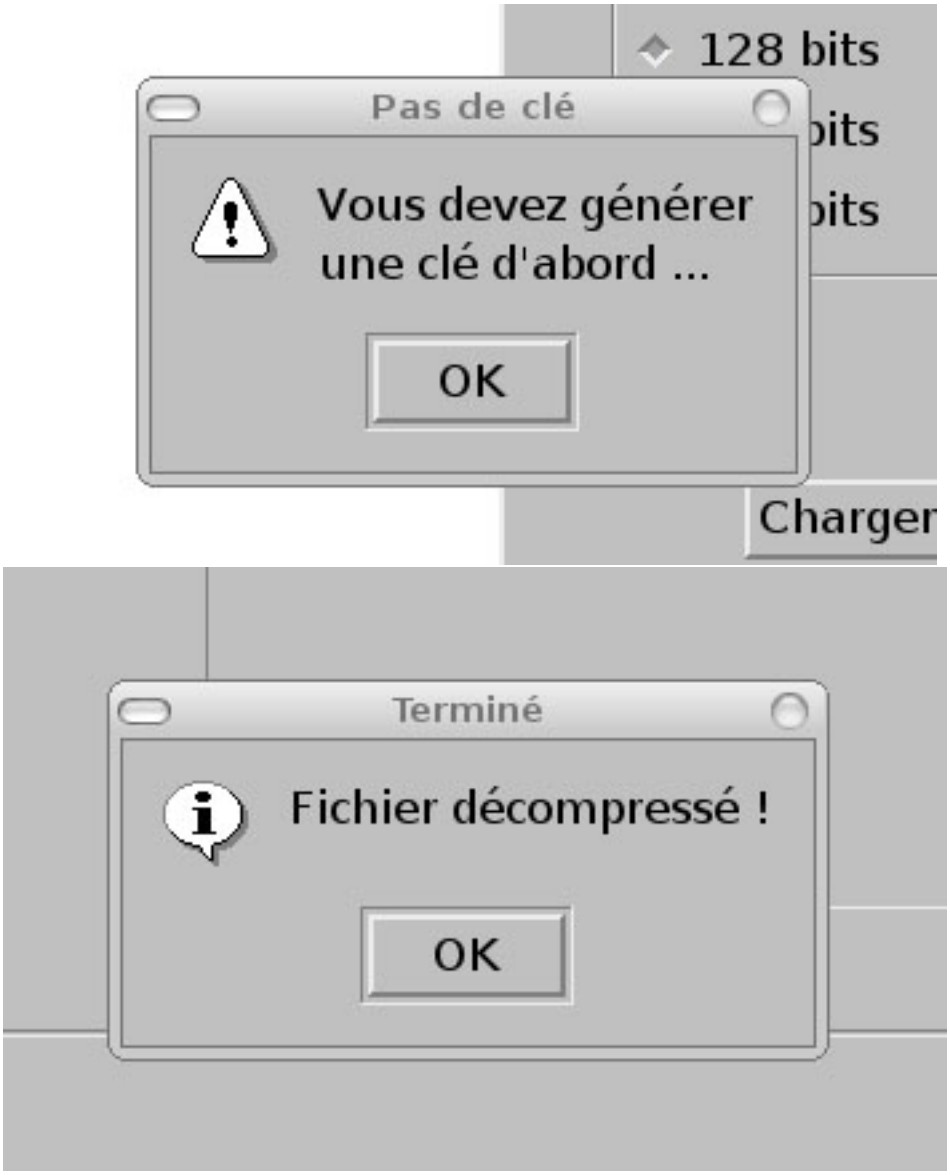
Chiffrement, qui permet de crypter des données, compression, stéganographie dans un fichier image (Bmp) et stéganographie dans un fichier audio (Wav) pour dissimuler des données (qui peuvent être auparavant cryptées et/ou compressées et enfin déchiffrement, pour rendre lisible un fichier préalablement chiffré.



Chaque onglet se compose de divers boutons ou zones de texte permettant à l'utilisateur d'accéder à toutes les fonctionnalités de SCube ! A chaque fois, il peut choisir un fichier à ouvrir (celui-ci apparaît alors en aperçu et peut y apporter des modifications avant de le chiffrer), puis, lorsqu'il lance la fonctionnalité de l'onglet sur lequel il se trouve, il dois alors choisir le nom du fichier de sortie via une nouvelle boîte de dialogue qui apparaît. Il dispose ensuite de son fichier crypté/compressé/caché qui viens d'être créé.



J'avais déjà commencé, mais j'ai cette fois rajouté des messages d'avertissement à chaque mauvaise manipulation afin de mieux guider l'utilisateur et d'éviter de provoquer des plantages.



4 Problèmes rencontrés

4.1 Sylvain Bellone

Ce problème aurait pu être évité, pourtant il m'a fait perdre plusieurs heures. Pour expliquer rapidement, j'ai fait quelques modifications sur mon ordinateur il y a peu de temps. Mauvaises apparemment puisque certains programmes ne fonctionnent plus correctement. Il s'agit d'un problème de librairies. J'ai préféré ne pas faire une réinstallation complète avant la soutenance, pensant

que cela ne me gênerait pas pour continuer le projet. Je me suis trompé. Résultat : une erreur de segmentation incompréhensible et impossible à trouver. C'est seulement après plusieurs heures que je me suis décidé à essayer sur une autre machine, et ça a marché. Bref, une perte de temps un peu dommage, surtout lorsqu'elle a lieu peu de temps avant la soutenance. Il y a ensuite tous les problèmes de compatibilité lors de l'intégration des différentes fonctions que j'ai déjà décrit plus haut. Nous pouvons aussi ajouter le manque de documentation sur la stéganographie, mais après tout il est normal de devoir effectuer des recherches assez poussée pour un projet comme celui-ci.

4.2 Julien Mao

Pour cette troisiéme soutenance il a fallu corriger les imperfections de la fonction de compressions afin d'obtenir une fonction sans erreurs. L'implémentation des fonctions de décompressions n'ont pas posé de réelles difficultés algorithmiques,il a simplement fallu faire attention aux éventuels problémes de positionnement lors de la saisie des caractéres dans le fichier à décompresser.
En revanche 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.

4.3 Dimitri Mérault

Pas de difficulté particulière, mis à part le temps. En effet, entre le projet et les autes matières, il devient de plus en plus difficile de gérer le temps. Quelques bugs ont été trouvé dans certaines fonctions, bugs qui ont été corrigé avec quelques difficultés.

4.4 Vincent Taront

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.

5 Travail à réaliser

Le travail qui nous reste à réalisé pour cette dernière soutenance sera, bien evidemment, de terminer ce projet !

6 Conclusion

Pour cette troisiéme soutenance les outils de compressions et de stéganographie ont été achevés. Lors de cette soutenance nous avons commencé à implémenter les différentes fonctions de déchiffrement.
Il ne nous reste plus qu'à finir d'implémenter les fonctions InvKeyExpansion et InvMixColumns et implémenter la fonction de compression dans l'interface graphique.