VCODE

Programme de chiffrage selon l'algorithme de Blaise de Vigenère

Manuel utilisateur

Auteur : 5t4n (alias aubnat)
smandon@hotmail.com

Contenu de la documentation "VCODE"

Documentation VCODE

Pré-requis

- Connaître l'utilisation de la ligne de commande de son système d'exploitation
- Etre conscient que tout ce qui se trouve sur un support source doit être effacé et réécrit avec des octets aléatoires au même emplacement afin de faire disparaître le fichier source, etc.
- Comprendre que l'algorithme de Blaise de Vigenère est d'autant plus sûr que la clef est aussi longue que le contenu à chiffrer (un idéal difficilement atteint, surtout sur de grands fichiers)

Présentation rapide

« VCODE » est un programme de chiffrage poly-alphabétique en ligne de commande basé sur l'algorithme de Blaise de Vigenère . Il substitue, en fonction d'une clef, les octets un à un d'un fichier donné en source. Blaise de Vigenère (1523-1596), gentilhomme français était un habitué de la Cour de Charles IX, puis d'Henri III. Il fut souvent chargé de missions diplomatiques comme envoyé du Roi de France à Rome, au Saint-Siège. Il semble que ce soit auprès des experts du Chiffre de la Cour papale qu'il découvre et se passionne pour l'art de la cryptographie.

En dehors de ses travaux alchimiques, il a laissé de nombreux ouvrages, notamment le "Traicté des Chiffres ou secrètes manières d'escrire".
Son célèbre tableau, appelé "Carré de Vigenère" a été jugé longtemps impénétrable par les spécialistes du Chiffre. Blaise de Vigenère demeure incontestablement un des pères de la cryptographie moderne.

A la différence de Vigenère, « **VCODE** » n'utilise pas les 26 caractères de l'alphabet mais la table ASCII de 256 caractères. Aussi, une simple lettre, telle que la lettre « A », peut avoir 256 représentations issues de la table ASCII. Un fichier chiffré par « **VCODE** » peut être constitué de plusieurs caractères non imprimables. Tous les types de fichier peuvent être chiffrés sans restriction de taille ou autre.

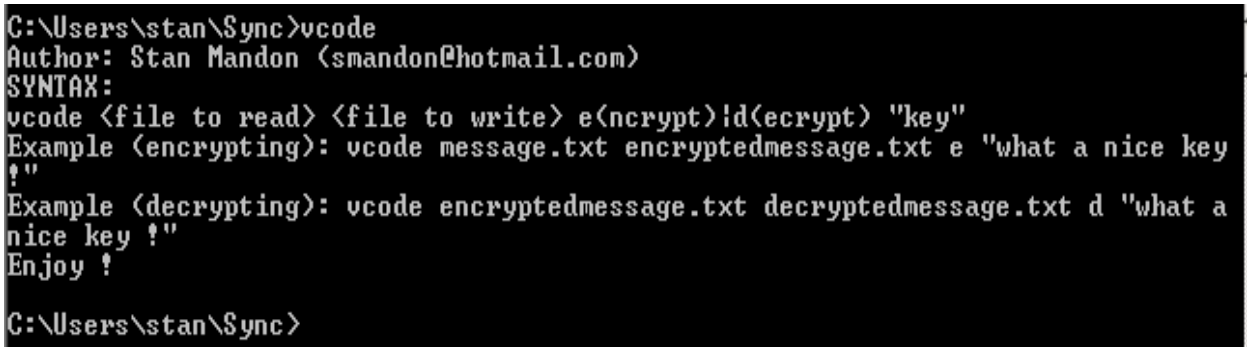
Auteur : 5t4n alias aubnat (smandon@hotmail.com)

La ligne de commande

Nous avons tellement l'habitude de travailler en « mode fenêtre » que nous en avons oublié l'essentiel. « **VCODE** » ne fonctionne pour l'heure qu'en ligne de commande. Nous comprendrons pourquoi ultérieurement. Aux débuts de l'informatique, tous les programmes un tant soit peu complexes fonctionnaient (et fonctionnent toujours d'ailleurs!) en ligne de commande. Il va donc falloir se familiariser avec une fenêtre « terminal » (sous Linux) ou « command » sous Windows. Une telle fenêtre vous demande de taper ce que vous souhaitez faire. Nous n'allons pas ici détailler une fenêtre de « terminal » ou de « command ». Juste intégrer qu'il va vous falloir taper la commande que vous souhaitez exécuter.

Lancer « VCODE »

Dans une fenêtre « **terminal** » ou « **command** » et à condition que vous soyiez dans le bon répertoire, la frappe de « **vcode** » suivie de la touche <**enter >** affiche un « **help** » sommaire. Ce « **help** » sommaire vous indique la syntaxe à adopter afin d'utiliser « **VCODE** » :



Ben ouais, « **vcode** » est en « angliche ». Mais on s'en fout, parce qu'on va comprendre comment s'en servir !

Exemple d'utilisation « au quotidien » : CHIFFRAGE

Vous venez de finir un superbe film de vacances en Italie, rempli de séquences très intimes et torrides auxquelles vous ne souhaitez pas que votre belle mère accède (aïe aïe aïe, imaginez le drame ! hihi). Ce film (ce fichier) vous l'avez appelé « `vacancesromaines.avi` ». **VCODE** va chiffrer votre film le plus simplement du monde, et cela, même si votre film fait 700 Méga ;)...

La lettre « `e` », troisième argument de la ligne de commande indique qu'on souhaite (e)ncrypter (chiffrer).

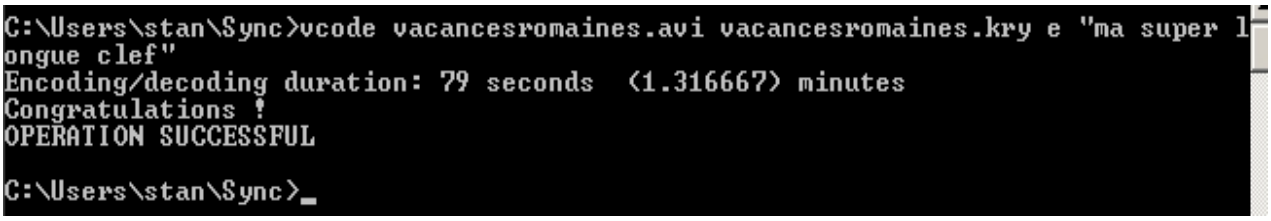
Voici comment procéder :

1)La ligne de commande à taper est la suivante :
`vcode vacancesromaines.avi vacancesromaines.kry e « votre clef qui doit être assez longue »`

2)Tapez **<enter>**

3) Patientez...
Laissez travailler **VCODE** (pour info, un fichier de 700 Mo sur une machine équipée d'un processeur Intel i5 prend environ 80 secondes à chiffrer ou à déchiffrer, quelle que soit la longueur de la clef).

A l'issue de ce traitement, vous devriez avoir un écran/message qui devrait ressembler à celui-ci :



En effet, dès que le chiffrage/déchiffrage est terminé, VCODE affiche la durée du traitement.

Remarques : *je vous conseille de donner à vos fichiers de destination une extension « significative ». Certes, **VCODE** acceptera n'importe quelle extension. A vous d'être cohérent avec vous-même ;). Nous avons utilisé ici .kry pour l'exemple.*

*Prenez garde au nom de fichier de destination. **VCODE** ne prévient pas en cas d'écrasement. Un écrasement est irréversible !*

Ce fichier chiffré `vacancesromaines.kry` est maintenant illisible par tous les players gratuits ou non du marché. Même si vous le renommez en .avi il restera inexploitable.

Il ne vous reste plus qu'à archiver ou à envoyer par mail ou dans votre cloud le fichier .kry et à en donner la clef à qui vous le souhaitez ;)

Exemple d'utilisation « au quotidien » : DÉCHIFFRAGE

C'est l'opération inverse de la précédente. Les manipulations sont identiques, mis à part le contenu de la ligne de commande.

Le déchiffrage de notre fichier `vacancesromaines.kry` va se faire tout aussi facilement que son chiffrage. Pour ce faire, nous n'avons qu'un caractère à changer. A savoir, le troisième argument de la ligne de commande : « `d` ». La lettre « `d` » pour (d)échiffrage , ainsi que les noms de fichier source et destination.

Notre ligne de commande pour le déchiffrage devient donc :
`vcode vacancesromaines.kry vacancesromaines.avi d « votre clef qui doit être assez longue »`

```
C:\Users\stan\Sync>vcode vacancesromaines.kry vacancesromaines.avi d "ma super l
ongue clef"
Encoding/decoding duration: 78 seconds  (1.300000) minutes
Congratulations !
OPERATION SUCCESSFUL

C:\Users\stan\Sync>
```

***ATTENTION**: prenez garde au nom de fichier de destination. **VCODE** ne prévient pas en cas d'écrasement. Un écrasement de fichier est irréversible !*

Exemple d'utilisation à 3 clefs

Nous allons maintenant jouer aux agents secrets (hihi). Supposons que nous ayons à donner un fichier à 3 destinataires différents mais que ce fichier ne puisse être déchiffré qu'en présence ou avec l'accord de ces 3 personnes. (Le nombre de destinataires importe peu. On pourra réaliser cette opération pour un nombre quelconque de destinataires...). Dans la présentation qui suit, nous avons utilisé un fichier texte pour des raisons de simplification. Il aurait très bien pu s'agir d'un fichier audio, vidéo ou tout autre .doc, .xls, odt, etc.

Le chiffrage

Nous allons donc chiffrer le fichier d'origine avec 3 clefs différentes. Chacune de ces clefs, en fait, ***une et une seule clef sera confiée à un et un seul destinataire***. Une seule personne connait ces 3 clefs : c'est vous !

Pour ce faire, nous allons séquentiellement chiffrer le fichier d'origine à 3 reprises avec 3 clefs différentes.

Afin d'éviter toute erreur, je vous conseille de vous faire un petit tableau du travail à effectuer.

Fichier d'origine	Clef	Fichier de destination	Destinataire de la clef
`Original.txt`	« clef1 »	`1.kry`	marmotte
`1.kry`	« clef2 »	`2.kry`	ledK
`2.kry`	« clef3 »	`Final.txt`	aubnat

Les trois ligne de commande à exécuter sont donc les suivantes :

```
vcode Original.txt 1.kry e « clef1 »  <enter>
vcode 1.kry 2.kry e « clef2 »  <enter>
vcode 2.kry Final.txt e « clef3 »  <enter>
```

Il ne vous reste plus qu'à transmettre le fichier « `Final.txt` » aux 3 destinataires (marmotte, LedK et aubnat dans notre exemple) ainsi que leur clef respective. Aucun d'entre eux ne peut désormais avoir connaissance du fichier d'origine sans la clef des autres.

***IMPORTANT**: l'ordre des clefs a son importance. En effet, **pour déchiffrer le fichier, il faudra STRICTEMENT procéder dans l'ordre inverse**.Toute erreur dans l'ordre des clefs ne fera que rechiffrer le fichier.*

Le déchiffrage

Les trois personnes sont maintenant réunies ou ont transmis leurs clefs respectives. Comme indiqué précédemment, il faut suivre strictement l'ordre inverse du chiffrage. Nous avons donc besoin dans l'ordre des clefs des destinataires aubnat, LedK et marmotte. Comme pour le chiffrage, nous allons faire un petit tableau afin d'éviter toute erreur.

Fichier d'origine	**Clef**	**Fichier de destination**
`Final.txt`	« clef3 »	`2.tmp`
`2.tmp`	« clef2 »	`1.tmp`
`1.tmp`	« clef1 »	`Original.txt`

Les 3 lignes de commandes à exécuter sont alors les suivantes :

```
vcode Final.txt 2.tmp d « clef3 »
vcode 2.tmp 1.tmp d « clef2 »
vcode 1.tmp Original.txt d « clef1 »
```

Et voilà ! Il ne reste plus qu'à consulter le fichier « `Original.txt` ».

Recommandations de sécurité

Quelles que soient les données que l'on souhaite chiffrer, il faut savoir que la protection totale n'existe pas. A l'image des crèmes solaires, l'écran total est constitué de vêtements.

Aussi, le ou les fichiers d'origine qui seront chiffrés devraient théoriquement disparaître du support (disque dur, usb, sd, etc.). De plus, une bonne sécurité serait de réécrire à plusieurs reprises des octets aux emplacements même des fichiers de départ. Cette manipulation est bien évidemment réservée à des utilisateurs avertis, soucieux de sécurité et peut-être un peu paranoïaques (hihi).

Bref, on trouve sur le net ce type de programme qui écrivent aléatoirement des octets ou des 0 (zéros) dans les secteurs du support de stockage.

Dans le cas de chiffrage à plusieurs clefs, il faudrait aussi, tout comme pour le fichier d'origine, les faire disparaître et écraser les données se trouvant à leur emplacement physique sur le support de stockage.

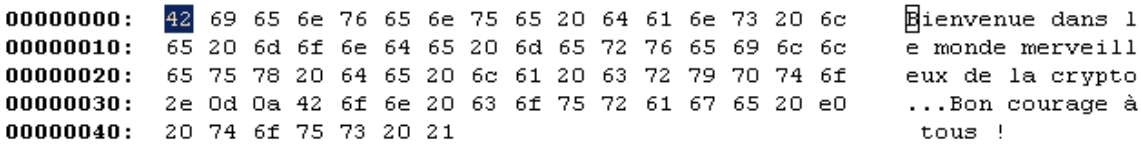
Annexe 1 : Détails d'un chiffrage à 3 clefs

Afin de bien comprendre, voici un exemple d'un fichier .txt chiffré à 3 reprises, que nous visualisons avec un éditeur hexadécimal.

Le fichier de départ « original.txt » contient le texte suivant :

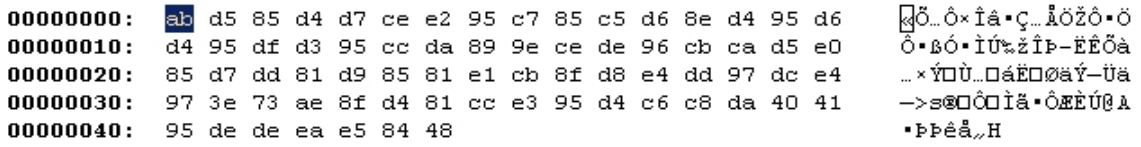
```
Bienvenue dans le monde merveilleux de la crypto.
Bon courage à tous !
```

En voici sa représentation dans un éditeur hexadécimal :



Clef utilisée : « `il fait beau aujourd'hui1` »

Le premier fichier chiffré 1.kry dans un éditeur hexadécimal :

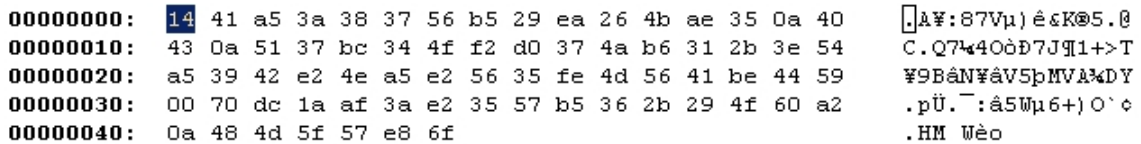


Vous pouvez constater que même les sauts de ligne ont été chiffrés. Et si vous ouvrez un logiciel tel que le bloc notes de Windows, le contenu ressemble à ça :

«Õ…Ô×Îâ•Ç…ÅÖŽÔ•ÖÔ•ßÓ•ÌÚ‰žÎÞ–ËÊÕà…×Ý⁄Ù…⁄áË„ØäÝ—Üä—
>s®Ô⁄Ìã•ÔÆÈÚ@A•ÞÞêå„H

Clef utilisée : « `il fait beau aujourd'hui2` »

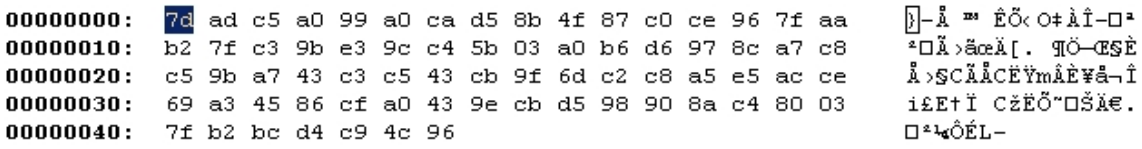
Le deuxième fichier chiffré 2.kry dans un éditeur hexadécimal :



Ici, le chiffrage précédent a été de nouveau chiffré avec une autre clef. Vous constaterez également que cette fois le fichier contient des caractères non imprimables, notamment aux offsets 14, 18 et 61 qui contiennent 0a et à l'offset 46 qui contient 00 (les offsets sont donné en décimal et en hexadécimal dans l'éditeur hexa.

Clef utilisée : « `il fait beau aujourd'hui3` »

Le dernier fichier chiffré final.txt dans un éditeur hexadécimal :



L'offset 60 (en décimal) contient 03. Dans le bloc notes ça donne :

}Å ™ ÊÕ‹O‡ÀÎ–□ª²□Ã›ãœÄ[# ¶Ö—Œ§ÈÅ›§CÃÅCËŸmÂÈ¥å¬Îi£E†Ï CžËÕ˜□ŠÄ€#□²¼ÔÉL–

La lettre « e » qui est présente 10 fois dans le fichier d'origine est chiffrée de 7 façon différentes (`C5 A0 8D B2 C4 97 8A`). Ceci est dû à la longueur de la clef de 25 caractères alors le contenu à chiffrer en comporte 70. Si la clef avait été aussi longue que le texte à chiffrer, il est certain que la lettre « e » présente 10 fois aurait été chiffrée de 10 manières différentes.

Annexe 2 : les algorithmes

La différence entre les deux algorithmes réside dans le signe « + » pour le chiffrage et le signe « - » pour le déchiffrage.

L'algorithme de chiffrage

```
for(  ; !feof(lire) ;  )
{
   if(indiceclef >= (lgclef)) indiceclef = 0;
   c = fgetc(lire) + clef[indiceclef];
   if(!feof(lire)) fputc((int)c, ecrire);
   indiceclef++;
}
```

L'algorithme de déchiffrage

```
for(  ; !feof(lire) ;  )
{
   if(indiceclef >= (lgclef)) indiceclef = 0;
   c = fgetc(lire) - clef[indiceclef];
   if(!feof(lire)) fputc((int)c, ecrire);
   indiceclef++;
}
```