NF05

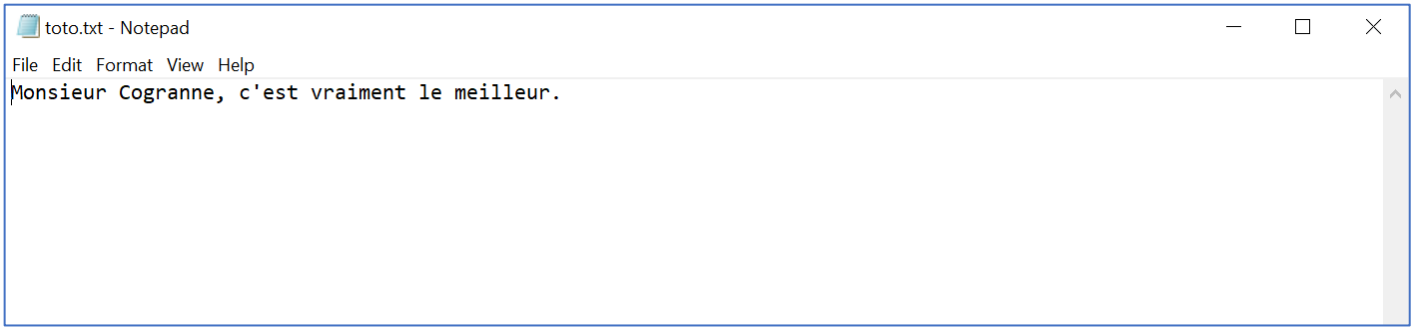
Initiation à la cryptographie symétrique

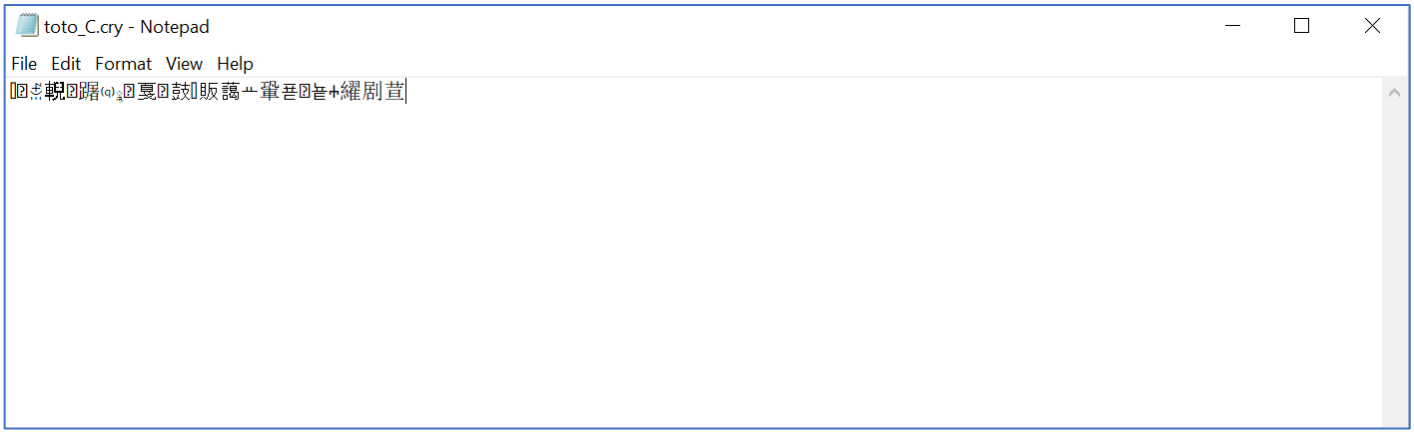
MAHFOUDH Tarek & PAPADAS Grigorios

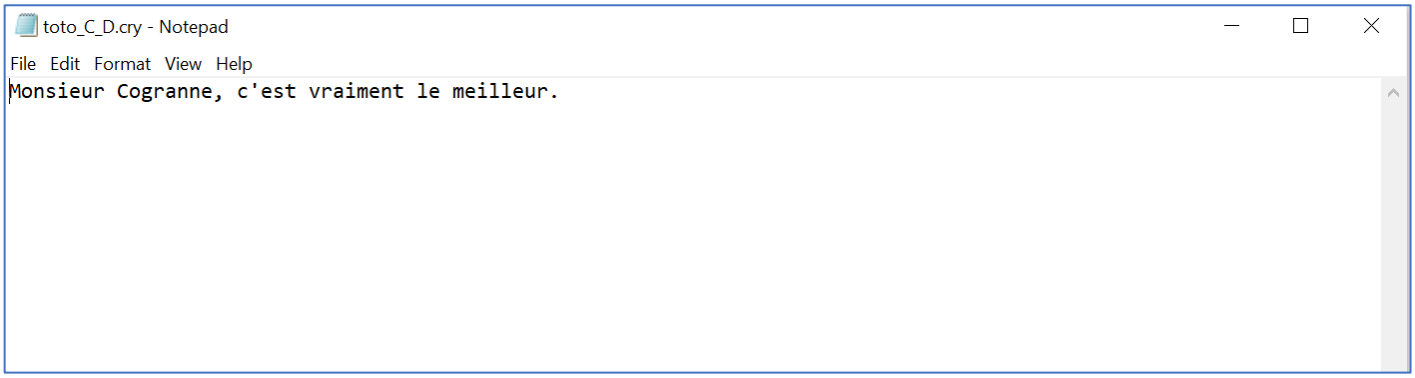
Automne

UNIVERSITÉ DE TECHNOLOGIE
TROYES

Projet Agamemnon : Initiation à la cryptographie symétrique







Une phrase en fichier dans un fichier texte, cryptée puis décryptée avec le mot de passe « HelloYou » et un nombre d'itération = 50.

Sommaire

Introduction

Dans le cadre de l'UE NF05, nous avons dû réaliser un projet portant sur l'initiation à la cryptographie symétrique, projet dans lequel nous nous sommes vus imposés quelques « méthodes » et manière de permettre la diffusion et confusion de notre message. Message qui, rappelons-le, pourra-être aussi bien une image, un fichier texte, une vidéo…

Dans l'optique de permettre à chacun, de comprendre la structure et le fonctionnement de notre programme, nous avons réalisé ce compte-rendu qui précisera la structure de l'algorithme, son fonctionnement, ses améliorations possibles, et enfin les problèmes rencontrés et les solutions apportées.

Dans le sujet, il nous est demandé de prendre un fichier, un mot de passe et un nombre d'itération (bonus).

De le crypter par bloc de 4 octets en 5 étapes (permutation des bits, permutation des octets, transformation affine, XOR et système linéaire appliqué au bloc).

Et ensuite de le décrypter en inversant les fonctions appliques au préalable.

L'utilisateur pourra au choix : chiffrer un fichier, déchiffrer un fichier chiffré au préalable par notre programme ou enfin choisir de voir une démo qui fera en sorte de faire les 2 successivement

Notre programme sert donc de démo et non de système de cryptologie fiable. **Il ne devra donc pas être utilisé pour le transfert de données sensibles.**

Nous verrons dans une première partie le mode d'emploi du programme, qu'est ce qui est demandé de l'utilisateur et à quoi doit s'attendre l'utilisateur.

Ensuite, pour mieux comprendre le fonctionnement et la structure de ce programme, nous allons l'étudier comme suit :

1. La structure de la fonction main ; les « préparations » nécessaires avant de se lancer dans les itérations et les étapes.
2. Les fonctions et procédures (on a seulement utilisé les procédures pour ne faire que des passages par adresse) qui ont permis de réaliser les 5 étapes demander. On entend par là les fonctions secondaires qui ne servent pas directement à chiffrer mais qui servent à assister dans le chiffrement. Celles-ci ont été écrites puisqu'elles allaient être utilisées plusieurs fois.
3. La fonction de chiffrement.
4. La fonction de déchiffrement.
5. L'étude de la fonction générant les sous-clés (et par conséquent les fonctions secondaires utilisées uniquement dans le cadre de cette génération).

Nous parlerons ensuite des problèmes rencontrés dans la réalisation du projet et des solutions que nous avons proposées et apportées pour y répondre.

Enfin, nous proposerons une conclusion à ce projet et à sa réalisation en exprimant notre ressenti sur ce que ce projet nous a apporté aussi bien en connaissances du langage C, qu'en algorithmie ou encore en cryptographie.

Mode d'emploi du programme :

Cette partie aura pour but de décrire tout ce que voit l'utilisateur une fois le programme lancé. De ce fait, ne devrons ici pas décrire les actions que réalise le programme, mais plutôt tout ce que voit et doit faire l'utilisateur.

En lançant le programme, on nous demande si on souhaite crypter, décrypter ou lancer la démo (qui sera utilisée lors de notre présentation).

```
Voulez-vous crypter ou decrypter un fichier ? Entrer 1 2 ou 3.
1 pour crypter
2 pour decrypter
3 pour lancer une demo (crypter et decrypter)
```

On rentre alors 1 ; 2 ou 3.

Ensuite apparait un message nous demandant d'entrer le nom du fichier (en incluant l'extension).

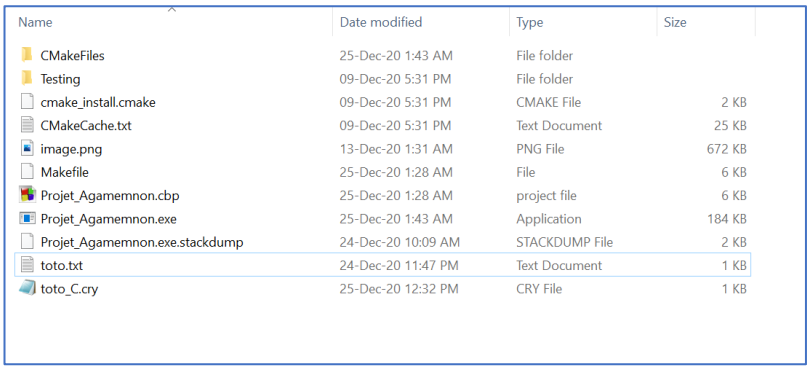
On rentre alors par exemple « toto.txt » ou encore « image.png » (etc.)

On nous demande ensuite le nombre d'itérations : on parle ici du N de la consigne, donc de combien de couches de chiffrement nous allons mettre en place. (Si N = 3. On chiffre le fichier une fois, on rechiffre le fichier déjà chiffre et on finit en chiffrant une dernière fois ce qu'on a obtenu au bout de N = 2).

Viens alors le tour du mot de passe : il nous est demande d'entrer un mot de passe de 8 caractères (on parlera de pourquoi on a choisi 8 plus tard et on verra qu'une amélioration est possible sous cet aspect-là).

Le programme nous montre alors les sous-clés utilises pour le chiffrement (une par itération) (générées à partir du mot de passe rentré)

```
Voulez-vous crypter ou decrypter un fichier ? Entrer 1 2 ou 3.
1 pour crypter
2 pour decrypter
3 pour lancer une demo (crypter et decrypter)
1
Entrer le nom du fichier (avec extension) :
toto.txt
Entrer le nombre d'iterations :
3
Entrez un mot de passe d'obligatoirement 8 caracteres:
cagranne
7s\0ww
yW{;F
M<U=Q
Thank you for using this program !
Press ENTER to exit.
```



Name	Date modified	Type	Size
CMakeFiles	25-Dec-20 1:43 AM	File folder	
Testing	09-Dec-20 5:31 PM	File folder	
cmake_install.cmake	09-Dec-20 5:31 PM	CMAKE File	2 KB
CMakeCache.txt	09-Dec-20 5:31 PM	Text Document	25 KB
image.png	13-Dec-20 1:31 AM	PNG File	672 KB
Makefile	25-Dec-20 1:28 AM	File	6 KB
Projet_Agamemnon.cbp	25-Dec-20 1:28 AM	project file	6 KB
Projet_Agamemnon.exe	25-Dec-20 1:43 AM	Application	184 KB
Projet_Agamemnon.exe.stackdump	24-Dec-20 10:09 AM	STACKDUMP File	2 KB
toto.txt	24-Dec-20 11:47 PM	Text Document	1 KB
toto_C.cry	25-Dec-20 12:32 PM	CRY File	1 KB

On voit l'apparition du fichier toto_C.cry généré par le programme.

Fonctionnement de l'algorithme :

Cette première partie a pour but de décrire l'algorithme, la logique appliquée pour chaque cas de notre sujet.

1. Comment s'organise le main (la structure du programme)

Dans cette partie nous aborderons 2 choses simultanément : nous parlerons de la structure du programme mais aussi des préparations qu'il a fallu faire avant de se lancer dans le chiffrement.

Dans la première partie, on demande d'abord à l'utilisateur ce qu'il souhaite faire (crypter/décrypter/démonstration) et son choix va être rentre dans une variable choix, on lui demande ensuite de rentrer au clavier le nom du fichier qui sera transformé (sécurité : on vérifie si le fichier existe bien, sinon erreur), le nombre d'itérations N (bonus) et un mot de passe de 8 caractères (sécurité : on vérifie si la longueur est bien celle indiquée).

A partir de là, l'utilisateur n'a plus rien à faire. Tout a été rentré. Le programme peut fonctionner avec les paramètres qu'il a à sa disposition.

On génère les N clés nécessaires au chiffrement (et on les affiche si on le souhaite).

On génère les noms de fichier _C.ext ou _D.ext (fichier crypté ou décrypté) en concaténant des chaines de caractères sans les créer car il nous les faut plus tard dans le chiffrement et/ou déchiffrement.

Enfin on met en place un switch dans lequel on rentre dans le chiffrement (choix = 1), le déchiffrement (choix = 2) ou la démonstration qui chiffrera le fichier puis déchiffrera le fichier chiffré (choix = 3).

2. Les fonction et procédures « secondaires » utilisées :

```
int test(char* nom);    //test file exists
long count(char nom[]); //counts bytes in file
void copieDansTab(char nom[], char temp[]); //copie le fichier dans un tab temp;
void copieDansFich(char nom[], char temp[]);    //copie le tab temp dans le fichier
void copyFileToFile(char nom_original[], char nom_copie[]);
void intToBin(int *tab, int nombre);
void binToInt(int *tab, int *nombre);
```

On va traiter et expliquer le fonctionnement et l'utilité des fonctions ci-dessus. Le temps estimé est en italique à la fin des explications des fonctions. (réflexion, mise en place et test compris)

On a ici :

- test : test pour voir si un fichier existe *(15 minutes)*
- count : donne la taille du fichier *(30 minutes)*
- copieDansTab : copie le fichier octet par octet dans un tableau alloué dynamiquement (grâce à count) *(45 minutes)*
- copieDansFich : copie un tableau dans un fichier. *(45 minutes)*
- copyFiletoFile : copie un fichier dans un autre fichier (copie simple). *(30 minutes)*
- intToBin : transforme un entier (ou caractère) en un tableau de taille 8 de 0 et de 1. *(30 minutes)*
- binToInt : transforme un tableau de taille 8 (de 0 et de 1) en un entier. *(30 minutes)*

Fonction int test :

On prend le nom (jusque-là supposé) du fichier et on l'ouvre en mode « r » (read). Si le pointeur est NULL on affiche alors une erreur : « Error opening file » (avec perror) et on fait return -1 pour directement arrêter la fonction main ; le nom de fichier rentré par l'utilisateur étant alors mauvais.

PS : cette fonction a été appelé en tout début de programme ; il est donc inutile de vérifier à chaque fois, entre deux fonctions si le fichier existe.

Fonction long count :

On retourne un long puisque notre programme peut traiter des fichiers extrêmement lourds qui dépasse les bornes du int (en théorie). On ouvre le fichier grâce au nom de fichier reçu en entrée ; et on compte le nombre d'octet en utilisant fseek (ptr, 0, SEEK_END) et ftell(ptr).

Procédure copieDansTab :

Ici on copie simplement le contenu du fichier dans un tableau, on traite chaque octet du fichier comme un caractère qu'on place dans un tableau de caractères.

Procédure copieDansFich :

Pareil que pour la procédure précédente mais on fait l'inverse : on récupère le contenu du tableau de caractères et on le copie dans un fichier (dont on aura mis le nom en entrée).

Procédure copyFiletoFile :

Ici on copie un fichier dans un autre, (code très similaire à celui proposé dans le cours de Mr. Arbaoui). On copie caractère par caractère le contenu du fichier d'origine.

Procédure intToBin :

On récupère un entier écrit dans la base décimal, on l'écrit en base binaire et on place chaque bit (entier 1 ou 0) dans un tableau [8] d'entiers.

Procédure binToInt :

On récupère un tableau [8] d'entiers écrit dans la base binaire, on l'écrit en base décimal dans un entier.

```
//Add bytes to have blocks of 4 bytes
void Add(char nom[]);    //padding procedure: rounds up number of bytes to 4
void Remove(char nom[]);  //detects the number of added 000000000 and removes them (number going from 0 to 2)
```

Enfin on a les procédures de padding, qui vont transformer notre fichier en un fichier qui puisse être regroupé en blocs de 4 octets. On a Add() qui va ajouter les octets nécessaires (en l'occurrence des 0 en tout début de fichier) et le Remove() qui va les enlever(PS : on ne doit pas garder un variable qui connaisse le nombre d'ajout : la procédure doit pouvoir détecter les octets ajoutes au préalable.)

Procédure Add :

On compte le nombre d'octet constituant le fichier et on cherche à l'arrondir à un multiple de 4. On ajoute alors les x octets nécessaires (x allant de 1 à 3) en tout début de fichier : l'octet x est égale à : « 00000000 ». *(1 heure et 45 minutes)*

Procédure Remove :

On cherche à inverser les effets du Add, cependant on ne sait pas combien d'octets ont été ajoutés. On cherche alors à détecter le nombre d'octets injectes au préalable avec l'indication que les octets sont égales à 0 et qu'il y en a entre 1 et 3. Une fois qu'on les a reconnus : on les écarte du fichier en les supprimant. On revient donc au fichier initial. *(1 heure)*

3. La fonction de chiffrement

Structure :

```
void encryption(char nomC[], unsigned char *keys[], int N){
    for(int i = 1; i <= N ;i++){
        step1_encryption(nomC, keys[i]);
        step2_encryption(nomC);
        step3_encryption(nomC);
        step4_encryption(nomC, keys[i]);
        step5_encryption(nomC);
    }
}
```

Ayant N itérations successives, et donc en quelque sorte N couches successives de chiffrement. On met une boucle allant de 1 à N (pas de 0 à N non compris pour les clés, on verra ça plus tard) et répétant donc l'ensemble des étapes de 1 à 5 dans l'ordre. Il est important de noter que pour chaque itération une différente sous-clé est utilisée.

Etape 1 : (ancienne et nouvelle version)

Ancienne version : On dispose de la sous-clé Ki (sous-clé de 8 caractères) correspondant à l'itération numéro i dans laquelle nous nous trouvons. On fait une opération pour avoir ce qu'on a appelle le poids de la sous-clé. L'opération est une série d'addition et de soustraction appliqué aux valeurs algébriques des caractères de la sous-clé. On a donc: poids = -key[0]+key[1]-key[3]+key[4]-key[5]+key[6]-key[7] (addition/soustraction alternée)

La substitution mono-alphabétique mise en place est alors dans une forme de code César avec un décalage égale à « poids » lettre de la table ASCII.

PS : poids peut être supérieur à 256 (ou inférieur à 0) mais le modulo est pris en compte pour rectifier ce débordement. *(2 heures)*

Amélioration : Par la suite (grâce aux indications de Mr. Cogranne) on a voulu améliorer notre fonction. Pour ce faire, nous avons utilisé ce qu'on appelle : **Key Scheduling Algorithm**. On utilise la sous-clé où chaque caractère est placé dans le tableau Karray en tant qu'entier. On fait de même avec les autres caractères jusqu'à ce que les 256 cases d'un tableau Karray soient remplies des 8 valeurs entières de la sous-clé.

Un autre tableau Sarray est rempli des valeurs de 0 à 255 dans l'ordre. Une fois ces deux tableaux remplis on utilise un algorithme de permutation pseudo-aléatoire dépendant de la sous-clé pour permuter les valeurs de Sarray suivant le tableau Karray. De cette façon on obtient notre table de permutation qui dépend de la clé.

Nous avons alors notre tableau de permutation.

Pour finir on remplace chaque caractère c par la valeur correspondante dans le tableau Sarray. *(1:30 heures)*

Etape 2 :

Pour l'étape 2, on met en place une permutation des octets, on a travaillé comme indiqué dans l'énoncé en bloc de 4. La permutation mise en place est la suivante :

1-> 2
2 ->4
3 ->1
4 ->3

On copie le bloc de 4 octets dans un tableau temp de taille 4. On procède ensuite à la réorganisation du bloc de 4 octets. Selon le réarrangement (présent à gauche) ; on réécrit notre tableau et copiant les éléments du tableau temp. *(30 minutes)*

Etape 3 :

Dans l'étape 3, nous a avons une transformation affine. On a donc une opération de la forme AX + B avec A : une matrice 8x8 ; X : notre octet décomposé en 8 bits donc une matrice 8x1 ; et B : une matrice 8x1.

On travaille octet par octet ; on met donc une boucle générale allant de 0 à la taille du fichier.

On commence par calculer le produit matriciel AX dans une boucle imbriquée dans une seconde boucle. On ajoute ensuite la matrice B grâce à une dernière boucle. On finit par ajouter le modulo 2 (très important) avant de transférer notre tableau de bits pour être transforme en caractère. *(3 heures)*

Etape 4 :

Dans l'étape 4, on prend nos blocs de 4 et on applique un fonction XOR (le signe ^ en C) avec la sous-clé Ki correspondant à l'itération i. Notre sous-clé étant de 8 octets et en traitant le fichier par bloc de 4 octets, on alterne : donc le premier bloc sera chiffré grâce à la première partie de la sous-clé, le deuxième bloc grâce à la deuxième moitié de la sous-clé, le troisième par la première et ainsi de suite. *(3 heures)*

Etape 5 :

Dans cette étape 5, le système linéaire nous est imposé dans l'énoncé du sujet. On travaille là encore en blocs et on n'a qu'à l'appliquer le système en passant par un tableau entier [4]. Ainsi on copie le bloc dans le tableau entier et on applique le système linéaire renseigné dans le sujet. Suite a la somme des différentes valeurs on applique un modulo 256. (Aussi disponible ci-dessous). *(2 heures)*

```
Z[0] = Y [0] + Y [1];

Z[1] = Y [0] + Y [1] + Y [2];

Z[2] = Y [1] + Y [2] + Y [3];

Z[3] = Y [2] + Y [3];
```

4. La fonction de déchiffrement

Structure :

```
void decryption(char nomD[], unsigned char *keys[], int N){
    for(int i = N; i >= 1 ;i--) {
        step5_decryption(nomD);
        step4_decryption(nomD, keys[i]);
        step3_decryption(nomD);
        step2_decryption(nomD);
        step1_decryption(nomD, keys[i]);
    }
}
```

Là encore, comme pour le chiffrement, la structure est très similaire. Sauf qu'on va aller de N à 1. Bien sûr, le nombre d'itérations est le même. Mais on décroît pour donner la bonne sous-clé a la bonne itération.

On fait aussi attention au fait qu'on commence à défaire l'étape 5, puis l'étape 4… jusqu'à l'étape 1.

Etape Inverse 1 : (ancienne et nouvelle version)

Ancienne version : Ici aussi, pour faire l'opération inverse ; on fait un décalage de –(poids) et non plus de poids. On rappelle que le poids correspond pour chaque itération i et donc pour chaque sous-clé associée Ki :

poids = -key[0]+key[1]-key[3]+key[4]-key[5]+key[6]-key[7]

- poids = - (-key[0]+key[1]-key[3]+key[4]-key[5]+key[6]-key[7])

Et donc en appliquant un code César avec décalage de poids dans un premier temps et un décalage de « -poids » dans un deuxième temps. On se retrouve avec le message clair. *(2 heures)*

Amélioration : Dans le cadre de l'amélioration de la fonction 1 on crée la même table de permutation utilisée pour chiffrer le fichier en utilisant le même processus (voir l'étape 1). On trouve ensuite simplement le caractère recherche dans le tableau correspondant. *(1:30 heures)*

Etape Inverse 2 :

Ici, il nous faut simplement trouver l'inverse de notre fonction de permutation créer auparavant.

On a alors le tableau ci-après.

1->3
2->1
3->4
4->2

Là encore on passe par un tableau temp[4] pour faire ces permutations exactement comme nous les avons réalisés dans l'étape 2 du chiffrement. *(30 minutes)*

Etape Inverse 3 :

Dans le sujet est aussi donné la transformation affine inverse. On conserve la même structure et le même code que pour l'étape 3, on change juste les matrices impliquées dans le calcul en remplaçant « l'originale » par l'inverse. *(3 heures)*

Etape Inverse 4 :

L'étape inverse de la 4 repose là encore sur le principe du XOR ; et ce principe est très simple. Montrons un exemple (on représentera l'opération par un ^) :

x ^ y = z et z ^ y = x

Donc pour retrouver x ; il suffit d'appliquer la même fonction avec exactement les mêmes paramètres qu'au départ. L'inverse de la fonction 4 est donc une copie exacte de la fonction 4. *(3 heures)*

Etape Inverse 5 :

On refait les mêmes instructions que l'étape 5 mais cette fois ci ; on applique ce système linéaire là. Suite a la somme des différentes valeurs on applique un modulo 256.

Au lieu d'avoir Y [i] en fonction de Z ; on a maintenant Z [i] en fonction de Y. *(2 heures)*

Y[0] = Z [0] - Z [2] +Z [3];

Y[1] = Z [2] - Z [3];

Y[2] = - Z [0] + Z [1];

Y[3] = Z [0] - Z [1] + Z [3];

5. La génération des sous-clés

```
void Decallage(int* decal ,unsigned char* tab_gauche, unsigned char* tab_droit, int i); //helps generate keys
int prochain(int i);    //helps generate keys and used in Decallage
void KeyGen(int N, const char password[], unsigned char ***keys[]); //generates keys
```

Pour générer ces sous-clés ; on va utiliser les fonctions ci-dessus.

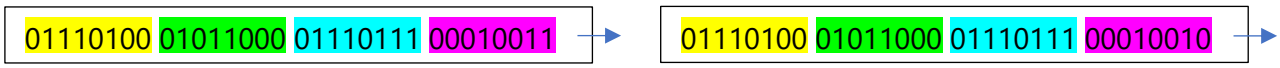
On verra dans un premier temps le concept et l'idée derrière la génération des sous-clés avant de plonger dans la manière de laquelle on a traité le problème.

Concept :

Pour la génération des sous-clés, il faut entrer un mot de passe de 8 caractères. On pourrait proposer une amélioration pour utiliser une clé de n'importe quelle taille, mais on verra que la génération des sous-clés n'est pas chose aisée si le nombre de caractère n'est pas une constante.

Le concept est simple, on coupe le mot de passe rentré par l'utilisateur en 2. La partie gauche (4 caractères – 32 bits) et la partie droite (4 caractères – 32 bits).

On fait un décalage à droite pour les 2 parties comme pour le schéma qui suit :



Donc tous mes bits vont connaitre un décalage à droite.

Quant à mes derniers bits (respectivement 1 pour la droite, 0 pour la gauche), ils vont être décalés du bit faible du 4^{e} et 8^{e} octet mais réapparaitront à en tant que bit fort dans le 1er et 5^{e} octet. Donc :

Et donc, à l'aide d'un seul décalage, on a de nouveaux octets et donc de nouveaux caractères. On recolle les 2 parties gauche et droite pour avoir notre sous-clé.

A l'aide de cette méthode, on peut générer 31 sous-clés toutes différentes du mot de passe. Si on insiste en mettant N > 31, au bout de la 32^{e} tentative, on tombera sur notre mot de passe (pas bon du tout). Au bout de la 33^{e} on retombera sur notre première sous-clé.

Une amélioration est possible pour augmenter le nombre de sous-clés si on utilisait une partie gauche et qu'on la concaténait avec les 30 autres combinaisons de la partie droite. Et pareillement avec la partie droite comme pivot. *(6-7 heures)*

Comment ?

Pour implémenter cette méthode dans notre programme en utilisant le langage C, on a dû utiliser les fonction décalage et prochain. En C, il est possible de « décaler » un caractère vers la droite (décaler les bits de x crans) en utilisant l'opérateur binaire « > ».

Se posent alors 2 problèmes majeurs. Le premier, on ne veut pas décaler un caractère mais une chaine de caractères (4 octets), or ce n'est pas possible grâce aux opérateurs binaires, il faudra le faire « manuellement ». Deuxièmement, l'utilisation de l'opérateur binaire « > » ne fera pas apparaitre la valeur

du bit faible à la place du bit fort après le décalage, on a toujours, automatiquement, un 0. Il est donc important d'évaluer la valeur du bit faible (1 ou 0) avant de décaler et de mettre en place une condition if pour rajouter un 1 si nécessaire après le décalage.

Face à ces problématiques, nous avons pu trouver le bon ordre d'instructions pour rendre possible cette génération de sous-clés.

Il s'agit alors d'une succession de décalage vers la droite entre les 4 caractères avec une condition if pour remettre le bit faible à l'emplacement du bit fort.

Problèmes et solutions :

Lors de la réalisation de ce projet, nous avons était confronté à plusieurs problèmes.

Le premier était bien évidement le problème de la génération des clés, qui a été sans nul doute la partie la plus complexe de ce projet. *(2 nuits blanches)*

Ensuite, bien que le sujet proposé par Mr. Cogranne fût très clair, il subsistait quand même quelques zones d'ombres comme les méthodes par lesquelles nous devions passer, je fais ici référence à la table de permutation qu'il fallait générer dans l'étape 1. La seule alternative qu'on avait trouvée c'est de faire un décalage de la table ASCII dont la valeur dépendait de la sous-clé utilisée. À la suite de l'aide apportée par Mr. Cogranne, qui nous a dernièrement proposé de jeter un coup d'œil au RC4, nous avons choisi d'utiliser un Key Scheduling Algorithm afin de créer une table de permutation avec 256 entrées distinctes qui dépend de la clé utilisée.

Pour finir, un des plus gros défis auxquelles on a dû faire face était la partie 3 : une erreur SIGSEGV (Signal Segmentation Violation), cette erreur est apparue puisqu'on a fait appel à une zone mémoire invalide. On a ensuite trouvé qu'il y avait une erreur dans l'initialisation du pointeur sur fichier. Mais le problème ne s'est pas arrêté là. En debuggant pour vérifier si la transformation donnait la bonne valeur : on remarque que si on rentre B dans un fichier .txt ; la sortie était de T. On fait alors un tableau Excel pour modéliser cette transformation affine. Et on utilise un convertisseur texte-binaire pour vérifier mes valeurs.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	Matrice H									octet X		HX		Matrice C		(HX+C) % 2
2	1	0	0	0	1	1	1	1		0		1		1		0
3	1	1	0	0	0	1	1	1		1		2		1		1
4	1	1	1	0	0	0	1	1		0		2		0		0
5	1	1	1	1	0	0	0	1		0		1		0		1
6	1	1	1	1	1	0	0	0		0		1		0		1
7	0	1	1	1	1	1	0	0		0		1		1		0
8	0	0	1	1	1	1	1	0		1		1		1		0
9	0	0	0	1	1	1	1	1		0		1		0		1
10																

Ici la transformation retourne 01011001 pour un caractère 01000010. Donc Y pour B. Il y a donc clairement un problème, après quelques minutes de débogage on s'est rendu compte que l'erreur de viens pas de l'ajout +C mais de la boucle que fais la multiplication entre les matrices H et X. On a décidé alors d'isoler les deux opérations sans les inclure dans la même boucle et le retour pour B est bien de Y au lieu du T qu'on avait eu précédemment. *(3 heures de plus)*

Enfin, il nous parait important d'indiquer que nous avons abandonné l'utilisation des procédures cypherChar et prochain(). L'une s'occuper de la génération de sous-clés ; elle a été remplacée puisqu'elle ne proposait qu'un maximum de 4 sous-clés (en incluant le mot de passe). L'autre servait dans une boucle for dans la procédure KeyGen() ; mais l'utilisation de la boucle étant inutile, par extension, cette procédure est aussi inutile.

Nous les avons gardées puisqu'on ne les considère pas comme déchets puisque tout est améliorable et que leur utilisation peut être bénéfiques dans d'autres projets ou pour une future amélioration de ce projet Agamemnon.

Conclusion :

A travers la réalisation du projet Agamemnon, on a fait énormément de progrès dans la maitrise du langage C, on a pu développer nos compétences dans le traitement de fichier, dans la manipulation de tableaux, dans l'interaction avec les bits… Mais bien que notre programme remplisse les conditions imposées dans le sujet mais aussi celles demandées en bonus. On se rend bien compte de la fragilité de notre système, le programme est améliorable dans tous ses aspects : que ce soit la génération de sous-clés, les méthodes utilisées (je pense par là aux étapes), bien qu'on ait améliorer l'étape 2 en changeant le « 2 à 2 » par une permutation entre les 4 octets du blocs (si on l'avait fait comme le sujet l'exigeait, l'étape 2 s'annulerait toutes les 2 itérations.) mais aussi pour l'étape 4 qui est trop facilement réversible sachant le XOR n'est pas totalement sécurisé. En raison d'un manque de temps et de maîtrise dans le maniement du langage C, ces améliorations n'ont pas toutes pu être réalisées.

Mais après tout ; tout peut-être éternellement améliorer et c'est bien là, la fatalité de la condition humaine : Toujours vouloir plus, toujours viser plus loin.

Mais bien que ce projet n'ait pas été une réussite sur le plan technique, ses bénéfices sont dans l'aspect éducatif qu'il a su proposer : « une initiation à la cryptographie symétrique ».

Nous avons appris énormément grâce à ce sujet ; tant sur les aspects techniques que logiques.

Enfin, je souhaiterais remercier Mr. Cogranne et Mr. Arbaoui, qui ont été là pour nous tout le long du semestre et qui ont su répondre à nos interrogations et à nos questions.

Références :

RC4 CIPHER SIMPLIFIED – Cryptography Home:
https://www.youtube.com/watch?v=1UP56WM4ook&t=123s&fbclid=IwAR3Y9OheEKkoKt8SSUOU92RIYXYFk_-dpzfBmb8YythNa0hz_xfL4Nx99F0

Lecture on DES by Christof Paar https://www.youtube.com/watch?v=kPBJIhpcZgE

Cryptographie Classique : chiffrement par transposition https://www.youtube.com/watch?v=nd0jfGcDUn8

Groupe symetrique : transpositions et cycles (Maths Adultes)
https://www.youtube.com/watch?v=Xi0IvDRD8ms

Groups of permutations Ludislau Fernandes https://www.youtube.com/watch?v=XPkabtWBKW0

Permutation Matrices – Lecture 9 by Jeffrey Chasnov
https://www.youtube.com/watch?v=d7AovBKeNMI&list=ULjMFv0MRmEuo&index=60

Feistel Cypher – Computerphile https://www.youtube.com/watch?v=FGhj3CGxl8I

Feistel Cipher Explained with example (Arabic) Mo3talogy
https://www.youtube.com/watch?v=w8_rloQMQ08

DES Key Creation - شرح بالعربي https://www.youtube.com/watch?v=8TWdL1C7DGY

Annexe : Le code commenté

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

int test(char* nom);     //test file exists
long count(char nom[]); //counts bytes in file
void copieDansTab(char nom[], char temp[]); //copie le fichier dans un tab temp;
void copieDansFich(char nom[], char temp[]);     //copie le tab temp dans le fichier
void copyFileToFile(char nom_original[], char nom_copie[]); //copier un fichier dans un
autre fichier
void charToBin(int *tab, char nombre);    //transpose un entier en base decimal dans un
tableau sous la forme binaire
void binToChar(int *tab, char *nombre);  //transpose un tableau en base binaire dans un
entier en base decimal

void Decallage(int* decal ,unsigned char* tab_gauche, unsigned char* tab_droit, int i);
//helps generate keys
int prochain(int i);     //helps generate keys and used in Decallage
void KeyGen(int N, const char password[], unsigned char ***keys[]); //generates keys
char cypherChar(char c, int i);    //not used anymore, helps cypher chars

//Add bytes to have blocks of 4 bytes
void Add(char nom[]);     //padding procedure: rounds up number of bytes to 4

//Encryption

void encryption(char nomC[], unsigned char *keys[], int N); //structure du chiffrement
(contient 1,2,3,4,5)
void step1_encryption(char nom[], unsigned char key[]);     //ancienne version du
chiffrement 1
void step1_encryption_amelioree(char nom[],unsigned char key[]);     //etape 1
chiffrement
void step2_encryption(char nom[]);  //etape 2 chiffrement
void step3_encryption(char nom[]);  //etape 3 chiffrement
void step4_encryption(char nom[],  unsigned char key[]);     //etape 4 chiffrement
void step5_encryption(char nom[]);  //etape 5 chiffrement


//Decryption

void decryption(char nomD[], unsigned char *keys[], int N); //structure du
dechiffrement (contient 1,2,3,4,5)
void step5_decryption(char nom[]);  //etape 5 dechiffrement
void step4_decryption(char nom[],  unsigned char key[]);     //etape 4 dechiffrement
void step3_decryption(char nom[]);  //etape 3 dechiffrement
void step2_decryption(char nom[]);  //etape 2 dechiffrement
void step1_decryption_amelioree(char nom[],unsigned char key[]);     //ancienne version
du dechiffrement 1
void step1_decryption(char nom[],unsigned char key[]);  //etape 1 dechiffrement

//take off added bytes to gain back the integrity of the file
void Remove(char nom[]);  //detects the number x of added 000000000 and removes them (x
going from 0 to 2)

///@brief C'est la fonction main : elle regroupe tout le programme qui va permettre à
```

```
l'utilisateur de crypter et/ou décrypter à sa guise et en rentrant les paramètres qui
sont nécessaires.
///La fonction main regroupe et fait appel à l'ensemble des fonctions et procédures.
C'est ici que l'utilisateur rentrera toutes les données.
int main() {

    printf("\nVoulez-vous crypter ou decrypter un fichier ? Entrer 1 2 ou 3.\n"
           "1 pour crypter\n"
           "2 pour decrypter\n"
           "3 pour lancer une demo (crypter et decrypter)"
           "\n");   //on demande a l'utilisateur le mode qu'il veut
    int choix = 0;
    scanf("%d", &choix);     //on recupere le choix
    if( (choix != 1) && (choix != 2) && (choix != 3) ) return -1;   //on test si son
choix est valide

    printf("Entrer le nom du fichier (avec extension) : \n");   //on demande le nom du
fichier
    char nom[255];
    scanf("%s", nom);   //on recupere le nom de fichier
    if (test(nom) == -1) return -1; //on test pour voir si le fichier existe, sinon on
arrete le programme

    int N;
    printf("Entrer le nombre d'iterations : \n");
    //on demande le nombre de couches de chiffrement ou dechiffrement a appliquer
    scanf("%d", &N);     //on recupere le nombre d'iterations

    char password[9];
    printf("Entrez un mot de passe d'obligatoirement 8 caracteres: \n");     //on
demande un mdp de 8 caracters
    scanf("%s", password);  //on le recupere
    if (strlen(password) != 8){
        printf("erreur : entrer 8 caracteres");
        return(-1);     //on arrete le programme si la longueur du mdp != 8
    }

    unsigned char **keys;
    keys = calloc(N+1, sizeof(char*));  //on initialise les N pointeurs pour sous-cles
    for(int i = 0; i <= N ; i++){
        keys[i] = calloc(strlen(password), sizeof(char));   //on donne une longueur de
8 aux sous-cles
    }

    KeyGen(N, password, keys);  //on genere les sous-cles
    //keys[0] est le mot de passe rentre par l'utilisateur
    //a partir de keys[1] a key[31](max) on a 31 sous-cles differentes l'une de l'autre
et differentes du mot
    //de passe.

//
//    for(int i = 1; i <= N ; i++){
//        for(int j = 0; j < 8; j++){
//            printf("%c", keys[i][j]);         //permet de lire les differentes sous-
cles generees
//        }
//        printf("\n");
//    }

/* a partir de ligne 99 a 114 on cree les noms de fichiers
```

```
 * donc pour par exemple : "xxxx".ext on aura
 * nomC = "xxxx"_C.cry
 * nomD = "xxxx"_D.ext */
    char nomC[255];
    char nomD[255];
    strcpy(nomC, nom);      //on commence par copier dans nomC
    strcpy(nomD, nom);      //et nomD
    int j = 0;
    char ext[10];   //chaine de caractere pour mettre l'extension du fichier d'origine
    while(nomC[j] != '.')    //on trouve la position du point
        j++;
    for (int k = 0 ; k < 5 ; k++){
        ext[k] = nom[j+k];  //on copie l'extension du fichier
    }
    nomC[j] = '\0'; //on coupe nomC a partir du point pour avoir nomC : "xxxx.ext" ->
"xxxx"
    strcat(nomC, "_C.cry"); //on concatene les 2 chaines de caracteres : "xxxx" ->
"xxxx_C.cry"
    nomD[j] = '\0'; //on coupe nomD a partir du point pour avoir nomD : "xxxx.ext" ->
"xxxx"
    strcat(nomD, "_D"); //on concatene les 2 chaines de caracteres : "xxxx" -> "xxxx_D"
    strcat(nomD, ext);  //et puis : "xxxx_D" -> "xxxx_D.ext"

    switch(choix){
        case 1 :
            copyFileToFile(nom, nomC);  //on copie le contenu du fichier d'origine dans
un autre fichier
            //on va chiffrer l'autre fichier qui est une copie du premier, pour quand
meme garder l'original
            Add(nomC);            //on fait le padding pour le fichier
            encryption(nomC, keys, N);  //on chiffre le fichier
            break;

        case 2 :
            copyFileToFile(nom, nomD);  //on copie le contenu du fichier chiffre dans
un autre fichier
            //on va dechiffrer l'autre fichier qui est une copie du chiffre, pour quand
meme garder la verion chifree
            decryption(nomD, keys, N);  //on dechiffre le fichier
            Remove(nomD);   //on enleve le padding pour le fichier
            break;

        case 3 :
            //dans le cas 3 : on refait exacetement le 1 et le 2 a la suite. voir en
haut pour les commentaires
            copyFileToFile(nom, nomC);
            Add(nomC);
            encryption(nomC, keys, N);
            copyFileToFile(nomC, nomD);
            decryption(nomD, keys, N);
            Remove(nomD);
            break;
        default:
            //si il y a erreur, on sort du programme en affichant le message d'erreur
            printf("\nError");
            return -1;
            break;
    }
    for(int i = 0; i <= N ; i++){
        free(keys[i]);   //libere les pointeurs
```

```
    }
    free(keys); //on libere le pointeur sur pointeurs


    printf("Thank you for using this program !\nPress ENTER to exit.");
    getchar();
    getchar();  //appuyez sur entrer pour sortir et terminer le programme
    return 0;
}




/// @brief Cette procédure est une structure de chiffrement qui utilise les 5 fonctions
de chiffrement
/// @brief Elle réalise les N chiffrements nécessaires pour crypter le message.
/// @param[in] N : Nombre d'itérations
/// @param[in] nomD[] : Le tableau contenant le fichier à chiffrer sous forme de
caractères
/// @param[in] *keys[] : Le tableau contenant les sous-clés
/// @param[out] nomD[] : Le fichier chiffré
/// @see step1_encryption_amelioree(), step2_encryption(), step3_encryption(),
step4_encryption(), step5_encryption()
void encryption(char nomC[], unsigned char *keys[], int N){
    //On met la boucle for qui fera les N iterations de 1 a 5
    for(int i = 1; i <= N ;i++){
//        step1_encryption(nomC, keys[i]);          //obsolete
        step1_encryption_amelioree(nomC, keys[i]);
        step2_encryption(nomC);
        step3_encryption(nomC);
        step4_encryption(nomC, keys[i]);
        step5_encryption(nomC);
    }
}

/// @brief Cette procédure est une structure de déchiffrement qui utilise les 5
fonctions de déchiffrement.
/// @brief Elle réalise les N déchiffrements nécessaires pour décrypter le message.
/// @param[in] N : Nombre d'itérations
/// @param[in] nomD[] : Le tableau contenant le fichier à déchiffrer sous forme de
caractères
/// @param[in] *keys[] : Le tableau contenant les sous-clés
/// @param[out] nomD[] : Le fichier déchiffré
/// @see step1_decryption_amelioree(), step2_decryption(), step3_decryption(),
step4_decryption(), step5_decryption()
void decryption(char nomD[], unsigned char *keys[], int N){
    //On met la boucle for qui fera les N iterations de 5 a 1 pour defaire les
chiffrements successifs
    for(int i = N; i >= 1 ;i--) {
        step5_decryption(nomD);
        step4_decryption(nomD, keys[i]);
        step3_decryption(nomD);
        step2_decryption(nomD);
        step1_decryption_amelioree(nomD, keys[i]);
//        step1_decryption(nomD, keys[i]);      //obsolete
    }
}
/// @brief Procédure de chiffrement 1
///
///
```

```
/// Cette procédure utilise les 8 caractères de la clé Ki afin de créer un poids de la
façon suivante :\n
/// poids = -key[0]+key[1]-key[3]+key[4]-key[5]+key[6]-key[7] (addition/soustraction
alternée).\n
/// Par la suite ce poids est ajouté à tous les caractères du tableau **nom[]**.\n
/// On applique alors un Code César de decalage **poids** à l'ensemble du fichier.\n
/// @param[in] nom[] : Tableau contenant le fichier à chiffrer sous forme de caractères
/// @param[in] key[] : Tableau contenant la sous-clé K d'indice i
/// @param[out] nom[] : Tableau contenant le fichier chiffré
/// @warning Cette procédure est première version non utilisée qui n'est plus utilisée,
voir  step1_encryption_amelioree()
/// @see count(), copieDansTab(), copieDansFich()
void step1_encryption(char nom[], unsigned char key[]){
    long length = count(nom);   //on recupere le nombre d'octet et donc de caracteres
    char *temp = calloc(length, sizeof(char));  //on prepare le tableau dans lequel on
va deverser le fichier
    copieDansTab(nom, temp);    //on copie le fichier dans le tableau
    int poids_mdp = -key[0]+key[1]-key[3]+key[4]-key[5]+key[6]-key[7];     //on definit
le poids
    for(int k = 0; k < length; k+=4){
        temp[k] += (char)poids_mdp; //le modulo se fais seul    //on fais le decalage
    }
    copieDansFich(nom, temp);   //on remet le teableau modifie dans le fichier
    free(temp); //on libere le pointeur sur tableau
}

/// @brief Procédure de déchiffrement 1
///
/// Cette procédure utilise les 8 caractères de la clé Ki afin de créer un poids de la
façon suivante :\n
/// poids = -key[0]+key[1]-key[3]+key[4]-key[5]+key[6]-key[7] (addition/soustraction
alternée).\n
/// Par la suite on soustrais le poids à tous les caractères du tableau **nom[]** afin
de les déchiffrer.\n
/// On inverse alors le Code César de decalage **poids** à l'ensemble du fichier.\n
/// @param[in] nom[] : Tableau contenant le fichier à déchiffrer sous forme de
caractères
/// @param[in] key[] : Tableau contenant la sous-clé K d'indice i
/// @param[out] nom[] : Tableau contenant le fichier déchiffré
/// @warning Cette procédure est première version non utilisée qui n'est plus utilisée,
voir step1_decryption_amelioree()
/// @see count(), copieDansTab(), copieDansFich()
void step1_decryption(char nom[],unsigned char key[]){
    long length = count(nom);   //on recupere le nombre d'octet et donc de caracteres
    char *temp = calloc(length, sizeof(char));  //on prepare le tableau dans lequel on
va deverser le fichier
    copieDansTab(nom, temp);    //on copie le tableau
    int poids_mdp = -key[0]+key[1]-key[3]+key[4]-key[5]+key[6]-key[7];  //on retrouve
le poids definit
    for(int k = 0; k < length; k+=4){
        temp[k] -= (char)poids_mdp; //on enleve le decalage
    }
    copieDansFich(nom, temp);   //on remet le tableau modifie dans le fichier
    free(temp); //on libere le pointeur sur tableau
}


/// @brief Procédure de chiffrement 1
///
///
```

```
/// Cette procédure utilise un Key Scheduling Algorithm pour créer une table de
permutation.\n
/// La table possède 256 entrées différentes.\n
/// Suivant la table on change les caractères de **nom[]**.\n
/// @param[in] nom[] : Tableau contenant le fichier à chiffrer sous forme de caractères
/// @param[in] key[] : Tableau contenant la sous-clé K d'indice i
/// @param[out] nom[] : Tableau contenant le fichier chiffré
/// @note * La table de permutation est différente pour chaque itération (elle dépends
de la sous clé).
/// @note * La fonction peut traiter tous les caractères de la table ASCII (avec ASCII
étendue)
/// @see count(), copieDansTab(), copieDansFich()
void step1_encryption_amelioree(char nom[],unsigned char key[]){
    long length = count(nom);   //on recupere le nombre d'octet et donc de caracteres
    char *temp = calloc(length, sizeof(char));  //on prepare le tableau dans lequel on
va deverser le fichier
    copieDansTab(nom, temp);    //on copie le fichier dans un tableau
    int Karray[256];
    int Sarray[256];
    for(int i=0;i<256;i++){

        int nombre=(int)key[i%8]; //Conversion du caractere key[i%8]  en un entier
        if(nombre<0)nombre=nombre+256; //si le nombre est >127 il devient negatif donc
on le ramene a un positif
        Karray[i]= nombre%256; //On l'affecte au tableau Karray

        Sarray[i]=i;//Initialisation du tableau Sarray
    }

    int j=0;
    for(int i=0;i<256;i++){
        j=(j+Sarray[i]+Karray[i])%256; // initialisation de j
        int transfer=Sarray[i];//permutation entre s[i] et s[j]
        Sarray[i]=Sarray[j];
        Sarray[j]=transfer;
    }


    for(int i=0;i<length;i++){
        int nombre=(int)temp[i];//Conversion du caractere en un entier
        if(nombre<0)nombre=nombre+256; //si le nombre est negatif on le ramene a un
nombre positif
        nombre=nombre%256;
        temp[i]=(char)Sarray[nombre]; // affectation de la nouvelle valeur suivant la
table de permutation Sarray
    }
    copieDansFich(nom, temp);   //on met le tableau modifie dans le tableau
}

/// @brief Procédure de déchiffrement 1
///
///
///  Cette procédure utilise un Key Scheduling Algorithm pour créer une table de
permutation (même table que step1_encryption_amelioree).\n
///  La table possède 256 entrées différentes.\n
///  Pour chaque caractère de **nom[]** on trouve son antécédent suivant la table afin
de le déchiffrer.\n
/// @param[in] nom[] : Tableau contenant le fichier à déchiffrer sous forme de
caractères
/// @param[in] key[] : Tableau contenant la sous-clé K d'indice i
```

```
/// @param[out] nom[] : Tableau contenant le fichier déchiffré
/// @note * La table de permutation est différente pour chaque itération (elle dépends de la sous clé).
/// @note * La procédure peut traiter tous les caractères de la table ASCII (avec ASCII étendue)
/// @see count(), copieDansTab(), copieDansFich()
void step1_decryption_amelioree(char nom[],unsigned char key[]){
    long length = count(nom);   //on recupere le nombre d'octet et donc de caracteres
    char *temp = calloc(length, sizeof(char));  //on prepare le tableau dans lequel on va deverser le fichier
    copieDansTab(nom, temp);    //on copie le tableau
    int Karray[256];
    int Sarray[256];
    for(int i=0;i<256;i++){

        int nombre=(int)key[i%8]; //Conversion du caractere key[i%8]  en un entier
        if(nombre<0)nombre=nombre+256; //si le nombre est >127 il devient negatif donc on le ramene a un positif
        Karray[i]= nombre%256; //On l'affecte au tableau Karray

        Sarray[i]=i;//Initialisation du tableau Sarray
    }

    int j=0;
    for(int i=0;i<256;i++){
        j=(j+Sarray[i]+Karray[i])%256; // initialisation de j
        int transfer=Sarray[i];//permutation entre s[i] et s[j]
        Sarray[i]=Sarray[j];
        Sarray[j]=transfer;
    }


    for(int i=0;i<length;i++){
        int nombre=(int)temp[i]; //Conversion du caractere en un entier
        if(nombre<0)nombre=nombre+256; //si le nombre est negatif on le ramene a un nombre positif
        int k;
        for(k=0;k<256;k++)if(Sarray[k]== nombre)break; //On recherche la valeur correspondante a la variable nombre dans le tableau de permutation
        temp[i]=(char)k; // affectation de valeur avant chiffrement suivant la table de permutation Sarray
    }
    copieDansFich(nom, temp);   //on met le tableau modifie dans le tableau
}

/* Crypter etape 2:
 * 1->2
 * 2->4
 * 3->1
 * 4->3
 */

/// @brief Procédure de chiffrement 2
///
///
/// Cette fonction traite le fichier par blocs de 4 octets et les permute de façon suivante :
/// * 1->2
/// * 2->4
/// * 3->1
```

```
/// * 4->3
/// @param[in] nom[] : Le tableau contenant le fichier à chiffrer sous forme de
caractères
/// @param[out] nom[] : Le fichier chiffré
/// @see count(), copieDansTab(), copieDansFich()
void step2_encryption(char nom[]){
    long length = count(nom);   //on recupere le nombre d'octet et donc de caracteres
    char *fichier = calloc(length, sizeof(char));   //on prepare le tableau dans lequel
on va deverser le fichier
    copieDansTab(nom, fichier); //on copie le tableau
    char temp[4];   //on intitaialise un tableau temporaire pour les permutations
    for(int i = 0; i < length ; i+=4){
        //on recree notre permutation qu'on trouve ci-dessus
        temp[0] = fichier[i+1];
        temp[1] = fichier[i+3];
        temp[2] = fichier[i];
        temp[3] = fichier[i+2];
        //on remet nos permutations dans le tableau
        fichier[i] = temp[0];
        fichier[i+1] = temp[1];
        fichier[i+2] = temp[2];
        fichier[i+3] = temp[3];
    }
    copieDansFich(nom, fichier);    //on met le tableau modifie dans le tableau
    free(fichier); //on libere le pointeur sur tableau

}

/* Decrypter etape 2:
 * 1->3
 * 2->1
 * 3->4
 * 4->2
 */

/// @brief Procédure de déchiffrement 2
///
///
/// Cette procédure traite le fichier par blocs de 4 octets et les permute de façon
suivante :
/// * 1->3
/// * 2->1
/// * 3->4
/// * 4->2
/// @param[in] nom[] : Le tableau contenant le fichier à déchiffrer sous forme de
caractères
/// @param[out] nom[] : Le fichier déchiffré
/// @see count(), copieDansTab(), copieDansFich()
void step2_decryption(char nom[]){
    long length = count(nom);   //on recupere le nombre d'octet et donc de caracteres
    char *fichier = calloc(length, sizeof(char));   //on prepare le tableau dans lequel
on va deverser le fichier
    copieDansTab(nom, fichier); //on copie le tableau
    char temp[4];   //on intitaialise un tableau temporaire pour les permutations
    for(int i = 0; i < length ; i+=4){
        //on recree notre permutation qu'on trouve ci-dessus
        temp[0] = fichier[i+2];
        temp[1] = fichier[i];
        temp[2] = fichier[i+3];
        temp[3] = fichier[i+1];
```

```
        //on remet nos permutations dans le tableau
        fichier[i] = temp[0];
        fichier[i+1] = temp[1];
        fichier[i+2] = temp[2];
        fichier[i+3] = temp[3];
    }
    copieDansFich(nom, fichier);    //on met le tableau modifie dans le tableau
    free(fichier); //on libere le pointeur sur tableau

}


/// @brief Procédure de chiffrement 3
///
///
///Cette fonction consiste en un transformation matricielle est de la forme
<b>Hx+C</b>, avec :\n
///\verbatim
///H = {1,0,0,0,1,1,1,1}        C = {1}
///    {1,1,0,0,0,1,1,1}            {1}
///    {1,1,1,0,0,0,1,1}            {0}
///    {1,1,1,1,0,0,0,1}            {0}
///    {1,1,1,1,1,0,0,0}            {0}
///    {0,1,1,1,1,1,0,0}            {1}
///    {0,0,1,1,1,1,1,0}            {1}
///    {0,0,0,1,1,1,1,1}            {0}\endverbatim
/// Le fichier est chiffré octet pas octet
/// @param[in] nom[] : Le tableau contenant le fichier à chiffrer sous forme de
caractères
/// @param[out] nom[] : Le fichier chiffré.
/// @see count(), copieDansTab(), copieDansFich(), charToBin(), binToChar()
void step3_encryption(char nom[]){
    //on met nos matrices pour le chiffrement
    int MAT[8][8] = {
            {1,0,0,0,1,1,1,1},
            {1,1,0,0,0,1,1,1},
            {1,1,1,0,0,0,1,1},
            {1,1,1,1,0,0,0,1},
            {1,1,1,1,1,0,0,0},
            {0,1,1,1,1,1,0,0},
            {0,0,1,1,1,1,1,0},
            {0,0,0,1,1,1,1,1},
    };
    int MATC[8] =
            {1,1,0,0,0,1,1,0};

    long length = count(nom);   //on recupere le nombre d'octet et donc de caracteres
    char *tableau = calloc(length, sizeof(char));   //on prepare le tableau dans lequel
on va deverser le fichier
    copieDansTab(nom, tableau); //on copie le tableau
    int entree[8];  //on initialise 2 tableaux, l'un pour contenir les bits d'entree et
    int sortie[8];  // l'autre les bits de sortie
    for(int i = 0 ; i < length ; i++){
        charToBin(entree, tableau[i]);   //on transforme le caractere en serie de 8
bits
        for(int j = 0; j < 8 ; j++){
            sortie[j] = 0;
            for(int k = 0 ; k < 8 ; k++){
                sortie[j] += MAT[j][k] * entree[k]; //on fais la multiplication
matricielle Hx
```

```
            }
            sortie[j] %= 2; //on fais le modulo 2 (on a affaire a des bits)
        }

        for(int j = 0 ; j < 8 ; j++){
            sortie[j] += MATC[j];   //on ajoute le C de : Hx + C
            sortie[j] %= 2; //on fais le modulo 2 (on a affaire a des bits)
        }

        binToChar(sortie, &tableau[i]);  //on retransforme la serie de bits recuperee
en un caractere
    }
    copieDansFich(nom, tableau);    //on met le tableau modifie dans le tableau
    free(tableau); //on libere le pointeur sur tableau

}
/// @brief Procédure de déchiffrement 3
///
///
/// Cette procédure consiste en une transformation matricielle inverse de
step3_encyption .
///La transformation matricielle est de la forme <b>H'x+C'</b>, avec :\n
///\verbatim
/// H' = {0,0,1,0,0,1,0,1}     C' = {1}
///      {1,0,0,1,0,0,1,0}          {0}
///      {0,1,0,0,1,0,0,1}          {1}
///      {1,0,1,0,0,1,0,0}          {0}
///      {0,1,0,1,0,0,1,0}          {0}
///      {0,0,1,0,1,0,0,1}          {0}
///      {1,0,0,1,0,1,0,0}          {0}
///      {0,1,0,0,1,0,1,0}          {0}
///\endverbatim
/// Le fichier est chiffré octet pas octet.
/// @param[in] nom[] : Le tableau contenant le fichier à déchiffrer sous forme de
caractères
/// @param[out] nom[] : Le fichier déchiffré
/// @see count(), copieDansTab(), copieDansFich(), charToBin(), binToChar()
void step3_decryption(char nom[]){
    //on met nos matrices pour le dechiffrement
    int MAT[8][8] = {
            {0,0,1,0,0,1,0,1},
            {1,0,0,1,0,0,1,0},
            {0,1,0,0,1,0,0,1},
            {1,0,1,0,0,1,0,0},
            {0,1,0,1,0,0,1,0},
            {0,0,1,0,1,0,0,1},
            {1,0,0,1,0,1,0,0},
            {0,1,0,0,1,0,1,0},
    };
    int MATC[8] =
            {1,0,1,0,0,0,0,0};

    long length = count(nom);   //on recupere le nombre d'octet et donc de caracteres
    char *tableau = calloc(length, sizeof(char));   //on prepare le tableau dans lequel
on va deverser le fichier
    copieDansTab(nom, tableau); //on copie le tableau
    int entree[8];  //on initialise 2 tableaux, l'un pour contenir les bits d'entree et
    int sortie[8];  // l'autre pour les bits de sortie
    for(int i = 0 ; i < length ; i++){
        charToBin(entree, tableau[i]);   //on transforme le caractere en serie de 8
```

```
bits
        for(int j = 0; j < 8 ; j++){
            sortie[j] = 0;
            for(int k = 0 ; k < 8 ; k++){
                sortie[j] += MAT[j][k] * entree[k]; //on fais la multiplication
matricielle H'y
            }
            sortie[j] %= 2; //on fais le modulo 2 (on a affaire a des bits)
        }
        for(int j = 0 ; j < 8 ; j++){
            sortie[j] += MATC[j];   //on ajoute le C de : H'y + C'
            sortie[j] %= 2; //on fais le modulo 2 (on a affaire a des bits)
        }
        binToChar(sortie, &tableau[i]);  //on retransforme la serie de bits recuperee
en un caractere
    }
    copieDansFich(nom, tableau);    //on met le tableau modifie dans le tableau
    free(tableau); //on libere le pointeur sur tableau

}
/// @brief Procédure de chiffrement 4
///
///
/// Dans cette procédure on applique la fonction XOR entre une octet et un caractère de
la sous-clé Ki
/// Le fichier est chiffré par blocs de 4 octets de sorte qu'au premier bloc on utilise
la première moitié de la sous-clé Ki et au deuxième bloc la deuxième moitié de Ki.
/// @param[in] nom[] : Tableau contenant le fichier à chiffrer sous forme de caractères
/// @param[in] key[] : Tableau contenant la sous-clé K d'indice i
/// @param[out] nom[] : Tableau contenant le fichier chiffré
/// @see count(), copieDansTab(), copieDansFich()
void step4_encryption(char nom[],  unsigned char key[]){
    long length = count(nom);   //on recupere le nombre d'octet et donc de caracteres
    char *fichier = calloc(length, sizeof(char));   //on prepare le tableau dans lequel
on va deverser le fichier
    copieDansTab(nom, fichier); //on copie le tableau
    for(int k = 0; k < length; k+=4){   //on travaille par bloc
        //on alterne : chaque bloc par multiple de 2
        //sera chiffre par une moitie du mot de pass
        //c'est une amelioration qu'on a apportee
        for(int i = 0; i < 4; i++){
            //on fais l'operation du XOR grace a l'operateur binaire ^
            fichier[k+i] = fichier[k+i] ^ key[(k+i) % 8];
        }
    }
    copieDansFich(nom, fichier);    //on met le tableau modifie dans le tableau
    free(fichier); //on libere le pointeur sur tableau
}

/// @brief Procédure de déchiffrement 4
///
///
/// La procédure inverse est exactement la même que celle de chiffrement puisqu'il
s'agit d'une fonction XOR.
/// @param[in] nom[] : Tableau contenant le fichier à déchiffrer sous forme de
caractères
/// @param[in] key[] : Tableau contenant la sous-clé K d'indice i
/// @param[out] nom[] : Tableau contenant le fichier déchiffré
/// @see step4_encryption(), count(), copieDansTab(), copieDansFich()
//La bijective est exactement pareil puisqu'il s'agit d'une operation XOR
```

```c
void step4_decryption(char nom[],  unsigned char key[]){
    long length = count(nom);   //on recupere le nombre d'octet et donc de caracteres
    char *fichier = calloc(length, sizeof(char));   //on prepare le tableau dans lequel
on va deverser le fichier
    copieDansTab(nom, fichier); //on copie le tableau
    for(int k = 0; k < length; k+=4){
        //l'operation XOR est sa meme bijective, aucunement
        //besoin de changer le code
        for(int i = 0; i < 4; i++){
            //on fais l'operation du XOR grace a l'operateur binaire ^
            fichier[k+i] = fichier[k+i] ^ key[(k+i)%8];
        }
    }
    copieDansFich(nom, fichier);    //on met le tableau modifie dans le tableau
    free(fichier); //on libere le pointeur sur tableau
}


/// @brief Procédure de chiffrement 5
///
///
///\verbatim On applique le système linéaire suivant :
///
///     Z[0] = Y[0] + Y[1]
///     z[1] = Y[0] + Y[1] + Y[2]
///     Z[2] = Y[1] + Y[2] + Y[3]
///     Z[3] = Y[2] + Y[3]\endverbatim
/// @param[in] nom[] : Le tableau contenant le fichier à chiffrer sous forme de
caractères
/// @param[out] nom[] : Le fichier chiffré
/// @note * Avant de stocker les résultats dans le fichier on applique modulo 256
/// @note * La procédure peut traiter tous les caractères de la table ASCII (avec ASCII
étendue)
/// @see count(), copieDansTab(), copieDansFich(), CharToBin(), binToChar()
void step5_encryption(char nom[]){
    int length = count(nom);    //on recupere le nombre d'octet et donc de caracteres
    char *tableau = calloc(length, sizeof(char));   //on prepare le tableau dans lequel
on va deverser le fichier
    copieDansTab(nom, tableau); //on copie le tableau
    int temp[4];  //on creer un tableau temporaire pour faire les permutations

    //on travaille par blocs de 4
    for(int i = 0 ; i < length ; i += 4){

        //on ecrit le systeme lineaire
        temp[0]=(int)tableau[i]+(int)tableau[i+1];
        temp[1]=(int)tableau[i]+(int)tableau[i+1]+(int)tableau[i+2];
        temp[2]=(int)tableau[i+1]+(int)tableau[i+2]+(int)tableau[i+3];
        temp[3]=(int)tableau[i+2]+(int)tableau[i+3];

        //on fait les permutations (toujours avec le modulo)
        tableau[i]=(char) (temp[0] % 256);
        tableau[i+1]=(char) (temp[1] % 256);
        tableau[i+2]=(char) (temp[2] % 256);
        tableau[i+3]=(char) (temp[3] % 256);
    }
    copieDansFich(nom, tableau);    //on met le tableau modifie dans le tableau
    free(tableau); //on libere le pointeur sur tableau

}
```

```
/// @brief Procédure de déchiffrement 5
///
///
///@verbatim On applique le système linéaire suivant :
///
///     Y[0] = Z[0] - Z[2] + Z[3]
///     Y[1] = Z[2] - Z[3]
///     Y[2] = - Z[0] + Z[1]
///     Y[3] = Z[0] - Z[1] + Z[3]\endverbatim
/// @param[in] nom[] : Le tableau contenant le fichier à déchiffrer sous forme de
caractères
/// @param[out] nom[] : Le fichier déchiffré
/// @note * Avant de stocker les résultats dans le fichier on applique modulo 256
/// @note * La procédure peut traiter tous les caractères de la table ASCII (avec ASCII
étendue)
/// @see count(), copieDansTab(), copieDansFich(), CharToBin(), binToChar()
void step5_decryption(char nom[]){
    int length = count(nom);    //on recupere le nombre d'octet et donc de caracteres
    char *tableau = calloc(length, sizeof(char));   //on prepare le tableau dans lequel
on va deverser le fichier
    copieDansTab(nom, tableau); //on copie le tableau
    int temp[4];    //on creer un tableau temporaire pour faire les permutations

    //on travaille par blocs de 4
    for(int i = 0 ; i < length ; i += 4){

        //on ecrit le systeme lineaire
        temp[0]=(int)tableau[i]-(int)tableau[i+2]+(int)tableau[i+3];
        temp[1]=(int)tableau[i+2]-(int)tableau[i+3];
        temp[2]=-(int)tableau[i]+(int)tableau[i+1];
        temp[3]=(int)tableau[i]-(int)tableau[i+1]+(int)tableau[i+3];

        //on fait les permutations (toujours avec le modulo)
        tableau[i]=(char) (temp[0] % 256);
        tableau[i+1]=(char) (temp[1] % 256);
        tableau[i+2]=(char) (temp[2] % 256);
        tableau[i+3]=(char) (temp[3] % 256);
    }
    copieDansFich(nom, tableau);    //on met le tableau modifie dans le tableau
    free(tableau); //on libere le pointeur sur tableau
}

/// @brief Cette fonction permet de tester l'existance du fichier choisi
/// @param *nom : Pointeur sur le fichier choisi
/// @return  0: Le fichier existe
/// @return -1: Erreur
/// @note une condition dans le main verifie si test rend -1 le main s'arrette et rend
-1
int test(char* nom){
    FILE *ptr;
    ptr = fopen(nom, "r");  //on ouvre le fichier en mode Read
    if( ptr == NULL )  {    //on verifie si le fichier existe
        perror ("Error opening file");  //si on 'n'arrive pas a ouvrir le fichier, on
affiche un message d'erreur
        return(-1); //une condition dans le main verifie si test rend -1 ; si c'est le
cas ; le main rend -1
    }
    fclose(ptr);
    free(ptr);  //on libere le pointeur sur fichier
```

```
    return 0;
}

/// @brief Cette fonction permet de compter le nombre d'octets (caractères) dans un
fichier
/// @param nom[] : Le nom fichier choisi
/// @return La longueur du fichier en octets
long count(char nom[])
{
    FILE *ptr;
    ptr = fopen(nom , "r"); //on ouvre le fichier en mode Read
    fseek(ptr, 0, SEEK_END);    //on se place a la fin
    long length  = ftell(ptr);  //on lit la longueur
    fclose(ptr);    //on ferme le fichier
    free(ptr);  //on libere le pointeur sur fichier
    return length;  //on retourne la longueur
}


/// @brief Cette procédure permet de copier un fichier dans un tableau
/// @param[in] nom[] :  Nom du fichier choisi
/// @param[out] temp[] : Tableau où on copie le fichier
/// @note **nom[]** sert aussi d'argument à la fonction **count** pour calculer la
taille du fichier
/// @see count()
void copieDansTab(char nom[], char temp[]){
    FILE *ptr;
    long length = count(nom);   //on recupere le nombre d'octet et donc de caracteres
    ptr = fopen(nom, "r");  //on ouvre le fichier en mode Read
    for(int i = 0; i < length ; i++)
    {
        //on copie chaque caractere dans un fichier grace a fgetc (on aurait pu faire
sans la longueur grace a EOF)
        temp[i] = fgetc(ptr);
    }
    fclose(ptr);    //on ferme le fichier
    free(ptr);  //on libere le pointeur sur fichier

}

/// @brief Cette procédure permet de copier un tableau dans un fichier
/// @param[in] nom[] :  Nom du fichier choisi
/// @param[in] temp[] : Tableau à copier
/// @note **nom[]** sert aussi d'argument à la fonction **count** pour calculer la
taille du fichier
/// @see count()
void copieDansFich(char nom[], char temp[]){
    long length = count(nom);   //on recupere le nombre d'octet et donc de caracteres
    FILE *ptr;
    ptr = fopen(nom , "w"); //on ouvre le fichier en mode Write
    for(int i = 0; i < length; i++){
        fputc(temp[i], ptr);
        //on met le caractere dans le fichier grace a fputc
        //on le fais pour tous les caracteres dans le tableau
    }
    fclose(ptr);    //on ferme le fichier
    free(ptr);  //on libere le pointeur sur fichier
}
```

```c
/// @brief Cette procédure permet de copier un fichier dans un autre fichier
/// @param[in] nom_original[] : Nom du fichier à copier
/// @param[out] nom_copie[] : Nom du fichier où on copie **nom_original[]**
void copyFileToFile(char nom_original[], char nom_copie[]){
    FILE *ptr;
    FILE *ptr_copie;
    ptr = fopen(nom_original, "r"); //on ouvre le fichier qu'on souhaite copier en Read
    ptr_copie = fopen(nom_copie, "w");  //on ouvre le fichier destination en mode Write
    int c;  //ce caractere servira d'intermediaire
    while( (c = fgetc(ptr)) !=  (EOF) ){    //tant que le caractere recupere n'est pas
la fin du fichier
        fputc(c, ptr_copie);    //copie le dans le fichier destination
    }
    //ferme les 2 fichiers
    fclose(ptr);
    fclose(ptr_copie);
    free(ptr);  //on libere le pointeur sur fichier
    free(ptr_copie);  //on libere le pointeur sur fichier
}



/// @brief Cette procédure permet d'ajouter des octets qui valent 0 afin que la
longueur du tableau soit un multiple de 4 (padding).
/// @param[in] nom[] : Nom du fichier choisi
/// @note Si la longueur du fichier n'est pas un multiple de 4, on ne pourra pas
traiter le fichier par bloc de 4 octets.
void Add(char nom[]){
    int add;    //variable dans laquelle on mettra ce qu'il faudra ajouter
    long length = count(nom);   //on recupere le nombre d'octet et donc de caracteres
    add = ( 4 - (length % 4) ) % 4; //on calcul le nombre d'ajouts a faire
    //on doit avoir un fichier de taille multile de 4
    if (add != 0)   //s'il faut ajouter quelque chose
    {
        char *temp = calloc(length, sizeof(char));   //on prepare le tableau dans
lequel on va deverser le fichier
        copieDansTab(nom, temp); //on copie le tableau
        FILE *ptr;
        ptr = fopen(nom, "w");  //ouvre le fichier en mode write
        for(int i = 0; i < add ; i++)                       //on ajoute les blocs de
0000000 necessaires
        {
            fprintf(ptr, "%c", 0);
        }
        for(int i = 0; i < length ; i++)              //on rajoute le reste
        {
            fputc(temp[i], ptr);    //on copie le reste du tableau dans le fichier
        }
        fclose(ptr);    //on ferme le fichier
        //on free l'espace memoire utilise
        free(temp);
        free(ptr);
    }
}

/// @brief Enlever les octets qui valent 0, ajoutés par Add, afin de retrouver le
fichier initial
/// @param[in] nom[] : Le nom du fichier
/// @see count()
void Remove(char nom[]){
```

```
    int add = 0;
    long length = count(nom);   //on recupere le nombre d'octet et donc de caracteres
    char *temp = calloc(length, sizeof(char));  //on prepare le tableau dans lequel on
va deverser le fichier
    FILE *ptr;
    ptr = fopen(nom, "r");  //on ouvre le fichier en mode Read
    char c;
    //ce bloc for sert a determiner le nombre de 0 qu'on a ajouter dans le padding
    for(int i = 1 ; i < 4 ; i++){
        c = fgetc(ptr);
        if(c == 0) add++;
    }
    fclose(ptr);    //on ferme le fichier
    if (add != 0){  //s'il y a eu ajout
        ptr = fopen(nom, "r");  //on ouvre le fichier en mode Read
        int i;
        for(i = 0 ; i < add ; i++){
            fgetc(ptr); //on ignore les octets precedement ajoutes
        }
        while(i < length){
            temp[i-add] = fgetc(ptr);   //on copie le reste dans le tableau temp
            i++;
        }
        fclose(ptr);    //on ferme le fichier
        fopen(nom, "w");    //on ouvre le fichier en mode Write
        for(int j = 0; j < length-add ; j++)
        {
            fputc(temp[j], ptr);        //on replace le tableau dans le fichier (sans
le fichier)
        }
        fclose(ptr);    //on ferme le fichier
        //on free l'espace memoire utilise
        free(temp);
        free(ptr);
    }
}

/// @brief Cette procédure permet de convertir un caractère de la représentation
binaire (8 bits) en représentation dans la table ASCII
/// @param[in] *tab : Tableau (8 bits) contenant un caractère en représentation binaire
/// @param[out] *nombre : Pointeur sur le caractère
void binToChar(int *tab, char *nombre){  //transforme un tableau pour une
representation binaire en un caractere
    int temp = 0;
    for(int i = 0 ; i < 8 ; i ++){
        temp += tab[7-i]*pow(2, i);
    }
    *nombre = (char)temp;
}
/// @brief Cette procédure permet de convertir un caractère exprimé dans la table ASCII
sous forme binaire (8 bits)
/// @param[in] nombre : Un caractère
/// @param[out] *tab :  Tableau de 8 bits contenant le caractère sous forme binaire
void charToBin(int *tab, char nombre) {  //transforme un caractere en un tableau pour
une representation binaire
    int temp = (int) nombre;
    int k;
    int i = 8;
    for (i = 7; i >= 0; i--)
    {
```

```
        k = temp >> i;

        if (k & 1)
            tab[7-i] = 1;
        else
            tab[7-i] = 0;
    }
}

/// @brief Cette procédure utilise un caractère **c** et un entier **i** afin de couper
le caractère en deux puis décaler de **i** places à gauche les bits la première partie
et de **i** places à droite les bits de la deuxième.
/// @brief La fonction finit par recoller les deux parties du caractère.
/// @param[in] c : Le caractère à chiffrer
/// @param[in] i : Le nombre de places à décaler les bits
/// @return Un nouveau caractère
/// @warning CypherChar était la première version de la génération de clés. Cette
fonction n'est plus utilisée
//cypherChar etait la premier version de generation de cles ; cette methode n'est plus
utilisee
char cypherChar(char c, int i){     //tu donnes un char et un nombre i et il permute i
fois gauche et droite
    //on coupe notre caractere en 2 et on decale de i positions vers la droite
    unsigned char c_gauche = c >> 4;
    for(int j = 0; j < i ; j++){
        if (c_gauche%2 == 1){
            c_gauche = c_gauche >> 1;
            c_gauche = c_gauche +pow(2, 3);
        }
        else{
            c_gauche = c_gauche >> 1;
        }
        //on viens de remettre le 1 ou 0
    }
    c_gauche = c_gauche << 4;

    //on prend la partie droite et on decale de i positions vers la droite
    unsigned char c_droit = c << 4;
    c_droit = c_droit >> 4;
    //on defini la moitie droite (en effacant la gauche)
    for(int j = 0; j < i ; j++){
        if (c_droit % 2 == 1) {
            c_droit = c_droit >> 1;
            c_droit = c_droit + pow(2, 3);
        }else {
            c_droit = c_droit >> 1;
        }
        //on viens de remettre le 1 ou 0
    }
    return c_gauche + c_droit;
}


/// @brief Cette fonction permet la génération des sous-clés à partir d'un mot de passe
saisi par l'utilisateur.
/// @brief Les sous-clés sont utilisées dans les fonctions 1 et 4.
/// @param[in] N : Nombre d'itérations
/// @param[in] password[] : Mot de passe saisi par l'utilisateur constitué de 8
caractères.
/// @param[out] ***keys[] : Tableau contenant les sous-clés générés (passage par
```

```
adresse).
/// @warning keys[0] = password ; Donc ne pas utiliser keys[0]
/// @see Decallage()
void KeyGen(int N, const char password[], unsigned char ***keys[]){
    int L = 8;  //on sait que la longueur de la sous-cle est de 8 caracteres (on la met
dans une variable pour peut-etre ameliorer plus tard)
    unsigned char *tab_gauche, *tab_droit;
    tab_gauche = calloc(L/2, sizeof(char)); //on prepare le gauche de la sous-cle
    tab_droit = calloc(L/2, sizeof(char));  //on prepare la partie droite de la sous-
cle
    int i = 1;
    unsigned char temp[8];  //on definit un tableau de caracteres temporaire de taille
8
    strcpy(keys[0], password);  //keys[0] est toujours le mot de passe
    //c'est pour ca que dans les boucles, on part de 1 a = N et pas de 0 a < N
    strcpy(temp, keys[0]);  //on place aussi le mot de passe dans le tableau temporaire
    for(int j = 0 ; j < 4 ;j++){
        tab_gauche[j] = temp[j];    //on definit la partie gauche
    }

    for(int j = 4; j < 8 ;j++){
        tab_droit[j-4] = temp[j];   //on definit la partie droite
    }
    int decal[2];   //on a une taille de 2 ; l'une pour la partie gauche, l'autre pour
la droite
    int decal_temp[2];  //on aura besoin de 2 tableaux de 2 qui vont s'update
    do{
        Decallage(decal,  tab_gauche, tab_droit, 3);    //on commence par decaler le
dernier octet (3) le quatrieme
        for(int r = 0; r < 2; r++){
            decal_temp[r] = decal[r];   //on copie le contenu de decal dans decal_temp
        }

        Decallage(decal, tab_gauche, tab_droit, 0); //on decale le suivant
        if (decal_temp[0] == 1){    //si dans le decalage precedent on a 1, on met 1 :
sinon 0 automatique
            tab_gauche[1]+= (int) pow(2, 7);    //+2^7 signifie 1 dans le bit fort
        }
        if (decal_temp[1] == 1){    //si dans le decalage precedent on a 1, on met 1 :
sinon 0 automatique
            tab_droit[1]+= (int) pow(2, 7);
        }
        for(int r = 0; r < 2; r++){
            decal_temp[r] = decal[r];   //on copie le contenu de decal dans decal_temp
        }

        Decallage(decal, tab_gauche, tab_droit, 1); //on refais un decalage, on place
le prochain dans le tableau decal
        if (decal_temp[0] == 1){    //on corrige le decalage relaif a cet octet
            tab_gauche[1]+= (int) pow(2, 7);
        }
        if (decal_temp[1] == 1){    //on corrige le decalage relaif a cet octet
            tab_droit[1]+= (int) pow(2, 7);
        }
        for(int r = 0; r < 2; r++){
            decal_temp[r] = decal[r];   //on copie le contenu de decal dans decal_temp
        }

        Decallage(decal, tab_gauche, tab_droit, 2); //on refais un decalage, on place
le prochain dans le tableau decal
```

```
        if (decal_temp[0] == 1){     //on corrige le decalage relaif a cet octet
            tab_gauche[2]+= (int) pow(2, 7);
        }
        if (decal_temp[1] == 1){     //on corrige le decalage relaif a cet octet
            tab_droit[2]+= (int) pow(2, 7);
        }

        if (decal[0] == 1){ //on corrige le premier decalage realise, celui du 4e octet
            tab_gauche[3]+= (int) pow(2, 7);
        }
        if (decal[1] == 1){ //on corrige le premier decalage realise, celui du 4e octet
            tab_droit[3]+= (int) pow(2, 7);
        }

        strcpy(keys[i], tab_gauche);     //on copie la premiere partie de la sous-cle
        strcat( keys[i], tab_droit);     //on concatene avec la deuxieme partie de cette
sous-cle
        //on a genere keys[i] la i-eme sous-cle
        i++;

    } while (i <= N);    //(tant que i <= N) on doit generer N sous cle avec i
commencant a 1 ; voir declaration ligne 683
    free(tab_gauche);
    free(tab_droit);     //on libere les pointeurs sur fichiers
}

/// @brief Cette procédure aide à la génération des clés.
/// @brief Elle sert à décaler les bits d'un octet et à sauvegarder la valeur du bit
faible (maintenant "disparu").
/// @param[in] *tab_gauche : la moitié gauche de la sous-clé
/// @param[in] *tab_droit : la moitié droite de la sous-clé
/// @param[in] i : le i -ème caractère des deux moitiés de sous-clés
/// @param[out] *decal : Le tableau qui donne les valeurs des 2 bits faibles qui sera
mis dans les bits du prochain octet
void Decallage(int *decal ,unsigned char* tab_gauche, unsigned char* tab_droit, int i){
    unsigned char c = tab_gauche[i];
    //pour la partie gauche
    if(c%2 == 1){    //si le dernier bit = 1
        tab_gauche[i] = tab_gauche[i] >> 1; //on decale
        decal[0] = 1;    //on prend la valeur 1
    }
    else {  //si le dernier bit = 0
        tab_gauche[i] = tab_gauche[i] >> 1; //decale
        decal[0] = 0;    //on prend la valeur 0
    }

    //pour la partie droite
    c = tab_droit[i];
    if(c%2 == 1){    //si le dernier bit = 1
        tab_droit[i] = tab_droit[i] >> 1;    //on decale
        decal[1] = 1;    //on prend la valeur 1
    }
    else {  //si le dernier bit = 0
        tab_droit[i] = tab_droit[i] >> 1;    //on decale
        decal[1] = 0;    //on prend la valeur 0
    }
}

/// @brief Cette fonction retourne la valeur du bit suivant.
///
```

```
///@brief Elle est nécessaire surtout si le bit donne en paramètre est le 4e (=3). Dans
ce cas-là on doit rendre le 1er bit (soit =0).
///
///@param i : La valeur du bit dont on souhaite trouver le prochain bit
///@return k : la valeur de bit suivant
///@warning Cette fonction n'est plus utilisée. On a renoncé à la mise en boucle du
décalage.
///@warning On n'a donc plus besoin de cette fonction puisque tout est entrée
"manuellement".
int prochain(int i){    //il donne le prochain octet : si i = 3 ; prochain = 0  on ne
l'utilise plus dans le programme
    //i est l'entree, k la sortie
    int k;
    switch(i){
        case 0:
            k = 1;  //si i = 0 ; k = 1
            break;
        case 1:
            k = 2;  //si i = 1  ; k = 2
            break;
        case 2:
            k = 3;  //si i = 2 ; k = 3
            break;
        case 3:
            k = 0;  //si i = 3 ; k = 0
            break;
        default:
            k= -1;  //defailt "impossible" sinon il y a erreur
            break;
    }
    return k;
}
```