NF05 – Rapport Projet

Version 1.0

Valentin Koeltgen, Ahmed Bouhafa

Automne

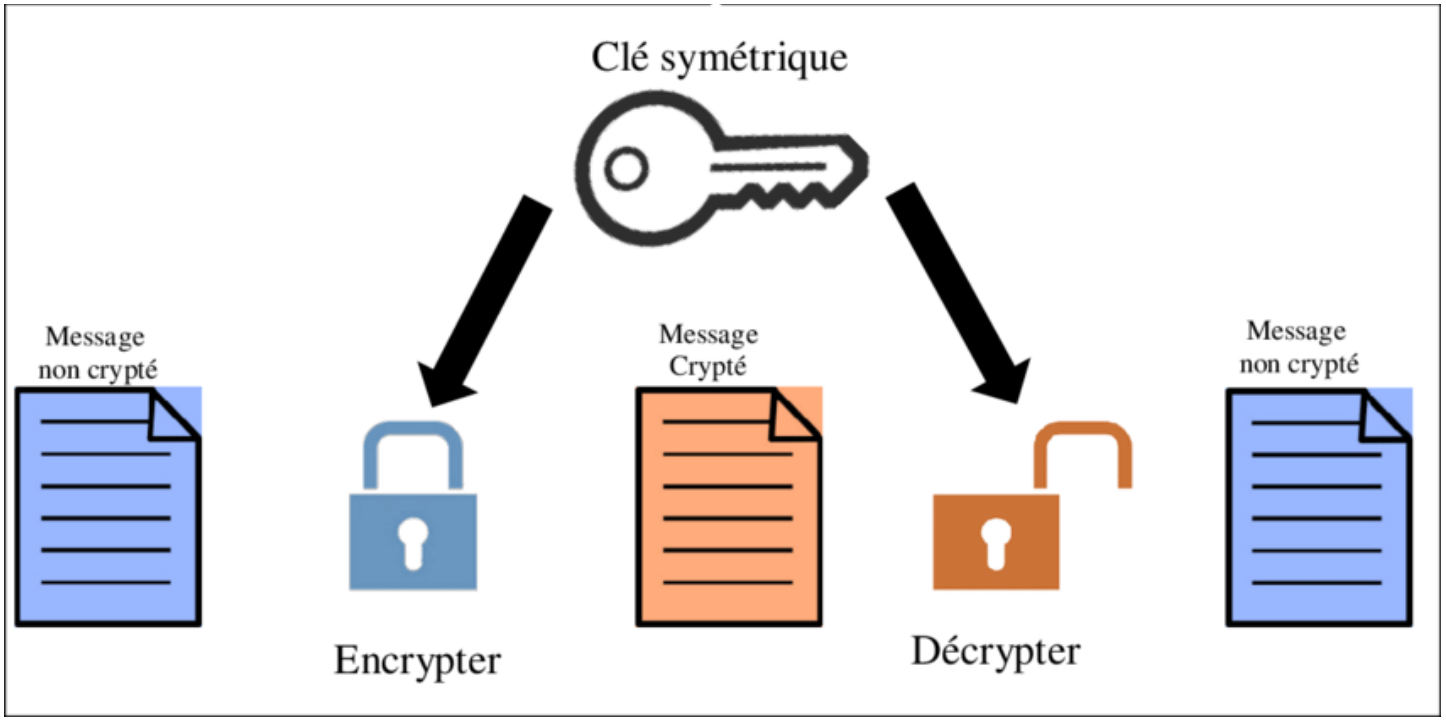
utt
UNIVERSITÉ DE TECHNOLOGIE
TROYES

Table des matières

Introduction

Le but de ce projet est de réaliser un programme permettant d'encrypter des fichiers quelconques avec une clé que l'on lui fournit (sous forme de chaîne de caractère) ainsi que de déchiffrer ces fichiers (avec cette même clé). Pour le cryptage, on utilise 5 méthodes différentes de cryptage symétrique (qui sont réversibles) que l'on va répéter N fois (N étant choisi par l'utilisateur) afin d'augmenter la sécurité du cryptage. Ce projet se fera purement en langage C avec une équipe de 2 développeurs.

Prenons l'exemple ci-dessous :



Un fichier est fourni au programme avec une « clé » et le nombre de répétitions N, le programme va encrypter le fichier puis stocker ce nouveau fichier au même endroit que le fichier fourni.

Inversement on peut demander au programme de décrypter un fichier en lui fournissant le fichier, la « clé » et le nombre de répétitions.

C'est pour cela que l'on appelle cette méthode la cryptographie symétrique, on va passer les mêmes paramètres en entrée pour crypter ou décrypter un fichier (à part le fait que pour décrypter, il faudra le fichier crypté, bien entendu).

Dans la prochaine partie nous vous décrieront les algorithmes utilisés pour créer ce programme, puis nous vous parlerons des problèmes que nous avons rencontré et des solutions que nous avons choisies. Nous vous donnerons ensuite un mode d'emploi expliquant l'utilisation de notre programme, puis nous finiront sur une conclusion synthétisant les différents aspects de ce projet et quelques points qui pourrait être améliorés.

La documentation en ligne est disponible au lien suivant : https://koeltv.github.io/Cryptographer/

Description des algorithmes

Cette partie sera divisé en 7 sous-parties, correspondant aux 7 fonctions principales implémentées dans le programme.

Voici le temps que l'on estime passer pour les différentes fonctions de notre code :

Fonctions de l'énoncé du projet	Temps passé (estimation avant-projet)
Fonction 1 : Permutation des caractères	30 min
Fonction 2 : Permutation 2-à-2	2h
Fonction 3 : Fonction matricielle	3h
Fonction 4 : XOR Logique	30min
Fonction 5 : Concaténation	1h
Fonctionnement global (sans les fonctions précédentes)	2h
Total	9h

Voici le temps dont on a eu besoin pour réaliser les différentes fonctions du code (valeur approximatives) :

Fonctions de l'énoncé du projet	Temps passé réel (valeur approximatives)
Fonction 1 : Permutation des caractères	50 min
Fonction 2 : Permutation 2-à-2	4h
Fonction 3 : Fonction matricielle	3h
Fonction 4 : XOR Logique	40min
Fonction 5 : Concaténation	1h30
Fonctionnement global (sans les fonctions précédentes)	3h
Total	13h

Fonction 0 : Lecture du fichier d'entrée

Cette fonction est assez simple, elle ne dépend donc d'aucune autre fonction[1].

Fonction principale : readFile()

Cette fonction prend en entrée un pointeur vers un entier pour stocker la taille du fichier. On va d'abord demander l'emplacement du fichier (relatif à l'emplacement du programme), puis on va tenter de l'ouvrir, si on échoue on arrête le programme et on renvoie l'erreur « Erreur: Impossible d'ouvrir le fichier <file> », <file> étant l'emplacement du fichier. Sinon on va d'abord chercher la taille du fichier, puis on créer dynamiquement un tableau pouvant stocker le fichier, on le lit et l'on stocke les valeurs caractère par caractère, puis on ferme le fichier. Une fois fini on renvoie en sortie un pointeur vers ce tableau.

Fonction 1 : Permutation des caractères

Cette fonction est assez simple, elle ne dépend donc d'aucune autre fonction.

Fonction principale : CharPermutation()

On prend en entrée un caractère (1 octet) que l'on va déplacer d'un certain nombre vers la droite sur la table ASCII, ce nombre va dépendre de la clé de chiffrement. Si l'on atteint la fin de la table ASCII (valeur > 255), on repart du début.

Pour inverser cette fonction lors du décryptage, on va faire exactement l'inverse, c'est-à-dire que l'on va déplacer le caractère d'un certain nombre vers la gauche (même nombre que la fonction ci-dessus) et si l'on atteint 0, on repart de la droite.

Fonction 2 : Permutation 2-à-2

Cette fonction dépend de plusieurs fonctions que nous allons vous présenter avant la fonction principale.

Lecture de la table de permutation : readTable()

On prend en entrée un lien vers le fichier à lire et un pointeur vers le tableau où seront stockés les valeurs, la fonction va lire le fichier et stocker les valeurs dans le tableau. Le fichier doit répondre aux critères suivants :

- Les entiers sont compris entre 0 et 255 inclus
- Chaque entier doit être différent
- Ils sont séparés par un caractère différent d'un nombre
- Il y a au moins 256 entiers (les entiers après les 256 premiers ne seront pas lus)

Si le fichier n'existe pas ou s'il ne répond pas aux critères, la fonction s'arrête et renvoie un tableau vide.

Fonction principale : permutationWithTable()

Dans cette fonction on va utiliser l'octet en entrée comme un indice qui va nous permettre de récupérer un autre octet ayant pour indice l'octet en entrée.

Pour inverser cette opération, on prend l'octet permuté que l'on recherche dans la table, et l'on prend son indice, c'est la fonction permutationWithTableReverse().

Fonction 3 : Fonction matricielle

Cette fonction dépend de plusieurs fonctions que nous allons vous présenter avant la fonction principale.

Décomposition de l'octet : ByteToBits()

Cette 1ère fonction prend en entrée un caractère (1 octet), le décompose en 8 bits grâce à la méthode des divisions successives et le renvoie en sortie dans un tableau d'entier.

Multiplication par la matrice : multiplyMatrices()

Ce programme prend en entrée un tableau d'entier à 2 dimensions et 1 à 1 dimension. C'est une version simplifiée d'un algorithme permettant la multiplication de 2 matrices puisque le tableau que l'on reçoit de la fonction précédente est toujours un tableau de 8 entiers.

On multiplie dans une boucle for toutes les valeurs d'une ligne de la 1ère matrice par la colonne de la 2ème puis on les stocke dans un tableau de la même taille (tableau de 8 entiers) en appliquant modulo 2 pour rester en binaire.

Recomposition de l'octet : BitsToByte()

Ici on recompose l'octet à partir d'un tableau de 8 bits en multipliant chaque bit par la bonne puissance de 2 puis en les additionnant.

Fonction principale : CharApplyMatrix()[1]

C'est ici que tout se passe, d'abord on décompose l'octet en bits avec ByteToBits(), puis on appelle multiplyMatrices() en fournissant une matrice fixée H et le tableau de la fonction précédente, puis on ajoute au résultat le vecteur C (élément par élément) toujours avec modulo 2 pour rester en binaire. Une fois que tous les calculs sont faits, on reforme l'octet avec BitsToByte().

Pour inverser cette fonction, on applique la même méthode mais en remplaçant la matrice H par la matrice H' et le vecteur C par le vecteur C', c'est la fonction CharApplyMatrixReverse().

[1] Les matrices H et H' ainsi que les vecteurs C et C' sont disponibles dans l'annexe

Fonction 4 : XOR Logique

Fonction principale : ApplyXOROnByte()

Dans cette fonction, on se sert de fonctions déjà présentées, ByteToBits() et BitsToByte() car on cherche à effectuer une opération bit par bit. On prend en entrée un octet que l'on décompose en bits puis l'on fait la somme avec un entier que l'on décompose similairement. Pour rester en binaire on applique modulo 2 puis on reforme l'octet avec BitsToByte().

Pour inverser la fonction il suffit de l'appliquer une seconde fois en passant les mêmes valeurs en paramètres.

Fonction 5 : Concaténation

Cette fonction est assez simple, elle ne dépend donc d'aucune autre fonction.

Fonction principale : concatenate()

Dans cette fonction, on prend un tableau de 4 octets, puis on leur applique les opérations suivantes :

$$\begin{cases} Z[0] = Y[0] + Y[1] \\ Z[1] = Y[0] + Y[1] + Y[2] \\ Z[2] = Y[1] + Y[2] + Y[3] \\ Z[3] = Y[2] + Y[3] \end{cases}$$ Y étant l'entrée et Z le résultat.

Pour inverser cette fonction, il suffit de renverser le système pour avoir Y en fonction de Z :

$$\begin{cases} Y[0] = Z[0] - Z[2] + Z[3] \\ Y[1] = Z[2] - Z[3] \\ Y[2] = Z[1] - Z[0] \\ Y[3] = Z[0] - Z[1] + Z[3] \end{cases}$$, Z étant l'entrée et Y le résultat.

Cette opération est effectuée par la fonction concatenateReverse().

Fonction 6 : Ecriture du nouveau fichier

Fonction principale : writeInFile()

Cette fonction permet d'écrire les octets encodés/décodés dans un fichier. L'emplacement et le nom du fichier dépendent du fichier lu au début du programme : le fichier sortant sera au même emplacement que celui d'entrée et le nom est le même avec « _encrypted » ou « _decrypted » à la fin. Par exemple **toto.txt** après encryptage deviendra **toto_encrypted.txt**.

Problèmes rencontrés et solutions adoptées

1. 29/10/20 : Apprendre à utiliser Git et GitHub à travers CLion

Solution : Suivi de tutoriels et expérimentation.

2. 30/10/20 : Créer une fonction capable de lire n'importe quel type de fichier au format binaire

Solution : Recherche dans la bibliothèque de fonction d'une fonction correspondante, fonction trouvée ; fopen (en mode lecture binaire) et fscanf.

3. 30/10/20 : Permettre un développement plus rapide sans avoir à taper les entrées à chaque lancement du programme et en ayant un fichier simple à comprendre.

Solution : Création de « link » et « encryptionKey » grâce à « #define », valeurs temporaires, ainsi que d'un fichier « test.txt » permettant d'avoir une représentation plus visuelle du résultat de la conversion.

4. 12/11/20 : Apprendre à utiliser Doxygen

Solution : Lecture de la documentation et recherche de tutoriel (voir Annexe 2).

5. 13/11/20 : Rendre la documentation disponible en ligne

Solution : Découverte de GitHub Pages

6. 21/11/20 : Créer un fichier de sortie basé sur le fichier d'entrée

Solution : Création de la fonction writeInFile() qui reprend le lien du fichier d'entrée et crée un nouveau fichier dont le nom et l'extension sont basés sur le fichier d'entrée (voir fonction writeInFile()).

7. 16/11/20 : Lire des caractères accentués

Solution : Les caractères accentuées se trouvent dans une version étendue de la table ASCII standard et ont des valeurs comprises entre 128 et 255 (compris), mais ne se lisent pas comme des caractères normaux. Chacun de ses caractères est lu comme une combinaison de 2 ou 3 caractères, qu'il est donc nécessaire de lire et d'analyser pour reformer le caractère original (voir fonction readNonStandardAscii()). Ce processus n'est nécessaire que lors de l'encryptage.

Cette fonction a été abandonnée car on s'est rendu compte qu'elle n'était plus nécessaire

Mode d’emploi

A l’ouverture du programme, celui-ci commencera par vous demander si vous voulez crypter ou décrypter un fichier, après quoi 4 ou 5 informations vous seront demandées dans cette ordre :

1. L’action à effectuer (encryptage/décryptage),
2. Le lien du fichier à encrypter, relatif à l’emplacement du programme (ex : ../fichier/toto.txt),
3. La clé d’encryptage, sous forme de chaîne de caractère (ex : Ceci est une cle d’encryptage),
4. Le nombre d’itérations voulues (plus d’itérations donne un meilleur encryptage mais cela est plus long),
5. L'emplacement du fichier contenant le tableau de permutation si celui-ci n'est pas au même emplacement que le programme.

Une fois toutes ces informations saisies, le programme va commencer le cryptage/décryptage. Lors de ce processus, une barre de progression sera affichée et actualisé tout au long de l’exécution pour vous tenir au courant de l’état de l’encryptage/décryptage. Quand il aura fini vous serez informé par le texte suivant « Termine ! Retrouvez le resultat a l'emplacement suivant: <emplacement> », <emplacement> étant l’emplacement ou a été enregistré le fichier.

En cas d’erreur, si le fichier à encrypter n’existe pas par exemple, le programme vous indiquera cette erreur et vous donnera la possibilité de la corriger (dans le cas d’un fichier non existant, il vous proposera d’entrer à nouveau un lien vers un fichier).

Une fois que vous avez trouvé votre fichier, au même emplacement que le fichier de départ (cet emplacement vous sera indiqué par le programme), vous pouvez fermer la fenêtre du programme sans problème en appuyant sur la touche entrée.

Attention ! Le programme fonctionne avec tout type de caractères, y compris les caractères accentués, néanmoins cette liste de caractère reste non exhaustive (limitée aux caractères de la table ASCII étendue, voir Annexe 3), aussi faites attention au caractères spécifiques « trop exotiques ». Même si ceux si n’entraîneront pas la fermeture du programme et vous serons signalés, ils provoqueront des modifications qui seront visibles lors du décodage, par exemple le(s) caractère(s) problématique(s) remplacés par des chaînes de caractères illisibles, comme ceci : « Ã§Ã© ».

Conclusion

Ce projet nous a permis d'apprendre beaucoup sur le développement en équipe, ainsi que sur l'utilisation de plateformes et logiciels utiles au développement, c'est-à-dire Git, GitHub et Doxyfile.

Il nous a permis également de mettre en pratique et d'approfondir nos connaissances du langage C et du processus de gestion d'un projet en général.

Pour améliorer notre programme, nous pensons qu'il serait intéressant de revoir certaines parties du programme pour le rendre plus performant et peut-être d'utiliser des méthodes de cryptage plus complexes, comme la méthode CBC (pour « Cipher Block Chaining ») ou CTR (« Counter »), contrairement à la méthode actuelle appelée ECB (« Electronic Code Book »), ce qui augmenterait la sécurité et/ou la rapidité de l'encryptage. Sur le long terme il serait peut-être également intéressant d'implémenter une interface graphique ou plus simplement plus d'interaction avec l'utilisateur lors de la génération du fichier crypté ou du décryptage.

Annexe

Annexe 1 : Matrices et vecteurs de la fonction 3

Matrice H et vecteur C :

$$\mathbf{H} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \quad \text{et} \quad \mathbf{c} = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

Matrice H’ et vecteur C’ :

$$\mathbf{H}' = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{pmatrix} \quad \text{et} \quad \mathbf{c}' = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Annexe 2 : Liens des solutions apportés

https://www.tutorialspoint.com/cprogramming/c_file_io.htm → utilisation d’un fichier externe

https://youtu.be/TtRn3HsOm1s → utilisation de Doxygen

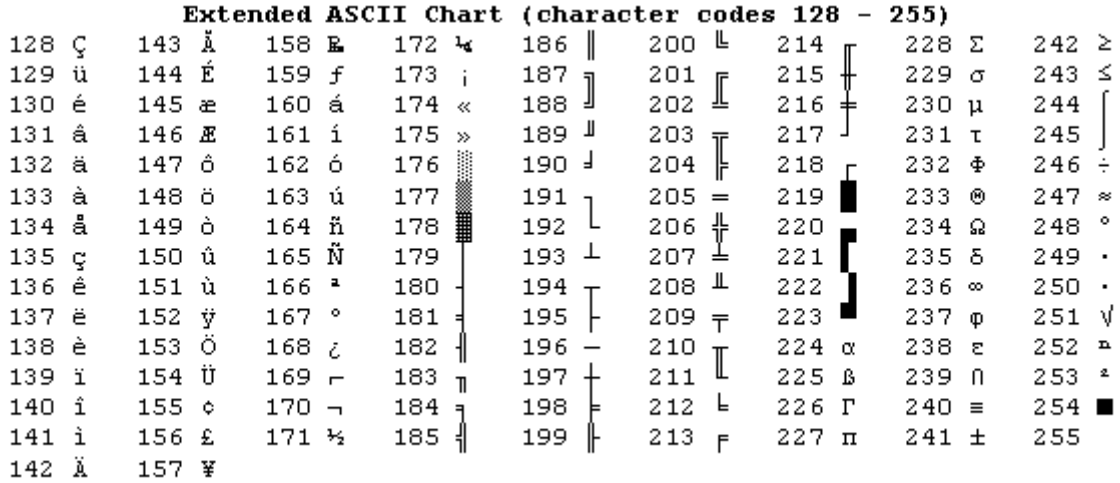
https://docs.github.com/en/free-pro-team@latest/github/working-with-github-pages/getting-started-with-github-pages → utilisation de GitHub Pages

Annexe 3 : Table ASCII Standard et table ASCII étendue

Table ASCII standard :

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Ajouts de la table ASCII étendue :



Annexe 4 : Code complet commenté du programme

Celui-ci est également disponible ici

```c
/**
 * @file main.c Fichier principal du programme
 *
 * Ce programme n'est constitué que du fichier main.c qui contient toutes les fonctions nécessaires à son fonctionnement
 */

#include <stdio.h>
#include <stdlib.h>

/**
 * Affiche une barre de progression
 * La fonction printProgress() affiche une barre de progression qui s'actualise à chaque appel
 * @date 17 Décembre 2020
 * @note inspiration: https://gist.github.com/amullins83/24b5ef48657c08c4005a8fab837b7499
 * @param currentIteration - numéro de l'itération en cours
 * @param total - nombre total d'itérations
 */
void printProgress(const int *currentIteration, const int *total){
    const char prefix[] = "Progression: [", suffix[] = "]";
    const unsigned char prefixLength = sizeof(prefix) - 1, suffixLength = sizeof(suffix) - 1;
    char *progressBar = calloc(prefixLength + 100 + suffixLength + 1, 1);

    for (int i = 0; prefix[i] != '\0'; i++) progressBar[i] = prefix[i];
    for (int i = 0; i < 100; i++) progressBar[prefixLength + i] = i < (100 - (((*total - *currentIteration) * 100) / *total)) ? '#' : ' ';
    for (int i = 0; suffix[i] != '\0'; i++) progressBar[prefixLength + 100 + i] = suffix[i];

    printf("\b\r%c[2K\r%s", 27, progressBar);
    fflush(stdout); free(progressBar);
}
```

```c
/**
 * Lit une chaîne de caractère
 * La fonction writeString() permet de lire une chaîne de caractère quelconque terminée par un retour à la ligne
 * @date 21 Novembre 2020
 * @note Cette fonction prend tout caractère visible à l'exception des retours à la ligne
 * @attention Cette fonction ne prendra pas un espace situé en début de chaîne (ex: " exemple" donnera "exemple")
 * @return addresse où est stocké la chaîne de caractère
 */
char *writeString(){
    int i = 0;
    char temp, *string = (char*) malloc(20 * sizeof(char)); //On part d'une chaîne de 20 caractères
    scanf(" %c", &temp);
    while (temp >= ' ' && temp <= '~'){
        if (i % 19 == 1 && i > 19) string = (char *) realloc (string, (i + 20) * sizeof(char)); //Si on dépasse 20 caractères, on ajoute un espace de 20 caractères à la chaîne
        string[i] = temp;
        scanf("%c", &temp);
        i++;
    } string[i] = '\0';
    return string;
}

/**
 * Lit le contenu d'un fichier
 * La fonction readFile() permet de lire le contenu d'un fichier quelconque dont on lui donne le lien (relatif au programme)
 * @date 29 Octobre 2020
 * @warning Si le fichier n'existe pas ou que la fonction n'arrive pas à le lire, elle affiche l'erreur suivante : "Erreur: Impossible d'ouvrir le fichier <fichier>", <fichier> étant le fichier que la fonction à essayer d'ouvrir
 * @note Dans le cas d'une erreur de lecture la fonction ne change pas size
 * @param fileLink - lien vers le fichier à lire
 * @param fileData - données récuperées du fichier
 * @param size - entier qui recevra la taille du fichier lu
 * ### Exemple
 * ~~~~~~~~~~~~~~.c
 *  //Ceci affiche le contenu du fichier test.txt à l'emplacement du programme
 *  unsigned char *file = NULL; int size = 0;
 *  readFile("./test.txt", &file, &size);
 *  if (size > 0) for (i = 0; i < size; i++) printf("%c", file[i]);
 * ~~~~~~~~~~~~~~
 */
void readFile(char *fileLink, unsigned char **fileData, int *size){
    FILE *sourceFile = NULL;
    if ((sourceFile = fopen(fileLink, "rb")) == NULL) fprintf(stderr, "Erreur: Impossible d'ouvrir le fichier %s", fileLink);
    else {
        fseek(sourceFile, 0L, SEEK_END); //Recherche de la fin du fichier
        *size = ftell(sourceFile); //Stockage de la taille
        rewind(sourceFile);
        unsigned char *temp = (unsigned char *) malloc(*size * sizeof(unsigned char)); //Création d'un tableau pour contenir les octets du fichier
        for (int i = 0; i < *size; i++) fscanf(sourceFile, "%c", &temp[i]);
        fclose(sourceFile);
        *fileData = temp;
    }
}
```

```c
/**
 * Ecrit dans un fichier
 * La fonction writeInFile() permet d'écrire des données dans un fichier basé sur le
fichier d'entrée et le mode choisi
 * @date 21 Novembre 2020
 * @note l'extension "_encrypted" ou "decrypted" sera ajouté à la fin du nom du fichier
selon le mode choisi
 * @warning Si un fichier du même nom existe déjà, il sera écrasé et remplacé par celui
généré par la fonction
 * @param data - données à écrire dans le fichier
 * @param size - quantité de données à écrire
 * @param sourceLink - lien vers le fichier source (utilisé pour le nom du fichier de
destination)
 * @param action - sélectionne l'extension "_encrypted" ou "_decrypted" (0 pour la 1ère,
toute autre valeur pour la 2ème)
 */
void writeInFile(unsigned char *data, const int *size, const char *sourceLink, const
unsigned char *action){
    int pointPosition=0, linkLenght=0;
    //Recherche taille du lien et position du point de l'extension de fichier
    while (sourceLink[linkLenght] != '\0'){
        if (sourceLink[linkLenght] == '.') pointPosition = linkLenght;
        linkLenght++;
    }
    char *destlink = (char *) malloc((linkLenght + 11) * sizeof(char));
    //Copie du début du lien jusqu'au point
    for (int i = 0; i < pointPosition; i++) destlink[i] = sourceLink[i];
    if (*action == 0) {
        char encrypted[] = "_encrypted";
        for (int j = 0; j <= 10; j++) destlink[pointPosition + j] = encrypted[j];
    } else {
        char decrypted[] = "_decrypted";
        for (int j = 0; j <= 10; j++) destlink[pointPosition + j] = decrypted[j];
    }
    //Ajout de l'extension du lien original
    for (int i = pointPosition; i <= linkLenght; i++) destlink[10 + i] = sourceLink[i];
    //Ecriture des données
    FILE *destinationFile = fopen(destlink, "wb");
    for (int i = 0; i < *size; i++) fprintf(destinationFile, "%c", data[i]);
    fclose(destinationFile);
    printf("\nTermine ! Retrouvez le resultat a l'emplacement suivant: %s", destlink);
    free(destlink);
}

/**
 * Permutation d'un octet
 * La fonction charPermutation() va prendre un octet et le décaler vers la droite dans la
table ASCII
 * @date 29 Octobre 2020
 * @note Si la valeur dépasse 255, on repart de 0
 * @param byte - caractère/octet à permuter, mis à jour suite à la permutation
 * @param key - valeur selon laquel byte sera décalé
 * ### Exemple
 * ~~~~~~~~~~~~~~.c
 *  unsigned char octet = 'a';
 *  printf("%c", charPermutation(byte, 2)); //Le résultat sera 'c'
 * ~~~~~~~~~~~~~~
 */
void charPermutation(unsigned char *byte, const unsigned char *key){
    *byte += *key % 256;
```

```c
}

/**
 * Inversion de la permutation d'un octet
 * La fonction charPermutationReverse() va prendre un octet et le décaler vers la gauche
dans la table ASCII
 * @date 29 Octobre 2020
 * @note Si la valeur atteint 0, on repart de 255
 * @param byte - caractère/octet à permuter, mis à jour suite à la permutation
 * @param key - valeur selon laquel byte sera décalé
 * ### Exemple
 * ~~~~~~~~~~~~~~.c
 *  unsigned char octet = 'c';
 *  printf("%c", charPermutationReverse(byte, 2)); //Le résultat sera 'a'
 * ~~~~~~~~~~~~~~
 */
void charPermutationReverse(unsigned char *byte, const unsigned char *key){
    *byte -= *key % 256;
}

/**
 * lit un fichier d'entier formaté
 * la fonction readTable lit un fichier d'entier formatée suivant les conditions
suivantes :
 *
 * - les entiers sont compris entre 0 et 255 inclus
 * - chaque entier doit être différent
 * - Ils sont séparés par un caractère différent d'un nombre
 * - il y a au moins 256 entiers (les entiers après les 256 premiers ne seront pas lus)
 *
 * @date 29 Décembre 2020
 * @attention Si le fichier n'est pas conforme aux points ci-dessus, la fonction s'arrête
et indique le problème
 * @warning Si le fichier n'existe pas la fonction affiche "Impossible d'ouvrir la table
de permutation" et s'arrête
 * @param link - Lien vers le fichier contenant la table de permutation
 * @param permutationTable - table de permutation mise à jour suite à la fonction
 */
void readTable(char *link, unsigned char **permutationTable){
    FILE *tableFile = NULL;
    if ((tableFile = fopen(link, "rb")) == NULL) printf("Impossible d'ouvrir la table de
permutation, ");
    else{
        fseek(tableFile, 0L, SEEK_END); //Recherche de la fin du fichier
        int size = ftell(tableFile); //Stockage de la taille
        rewind(tableFile);

        if (size >= 913){
            *permutationTable = (unsigned char *) calloc(256, sizeof(unsigned char));
            int i = 0, positionInFile = 0;
            while (i < 256 && *permutationTable != NULL){
                char temp;
                fscanf(tableFile, "%c", &temp); positionInFile++;
                while (temp >= '0' && temp <= '9' && positionInFile <= size){ //Lecture
et assemblage des chiffres
                    (*permutationTable)[i] = (*permutationTable)[i] * 10 + temp - '0';
                    fscanf(tableFile, "%c", &temp); positionInFile++;
                }
                for (int j = 0; j < i; j++){ //Si valeur dupliquée ou sortant de
l'intervalle
                    if ((*permutationTable)[i] == (*permutationTable)[j]){
```

```
                        printf("L'entier n%d et/ou l'entier n%d est/sont duplique(s) dans
la table ou sort(ent) de l'intervalle [0, 255], ", i, j);
                        free(*permutationTable); *permutationTable = NULL;
                        break;
                    }
                } i++;
            }
        } else printf("La table de permutation est incomplete ou mal formatee, "); //Il
manque des valeurs
        fclose(tableFile);
    }
}

/**
 * Permutation d'un octet selon une table de permutation
 * La fonction permutationWithTable() prend un octet et le modifie en prenant la valeur à
la position donnée par l'octet dans une table de permutation fournie par un fichier
externe nommé "permutation.txt"
 * @date 29 Décembre 2020
 * @warning le fichier "permutation.txt" doit être ua même emplacement que le programme,
sans quoi on obtient une erreur
 * @param byte - caractère/octet à permuter, mis à jour à la suite de la permutation
 */
void permutationWithTable(unsigned char *byte, const unsigned char *permutationTable){
    *byte = permutationTable[*byte];
}

/**
 * Inversion de la permutation d'un octet selon une table de permutation
 * La fonction permutationWithTableReverse() prend un octet et le modifie en prenant
l'indice associé à la valeur donnée par l'octet dans une table de permutation fournie par
un fichier externe nommé "permutation.txt"
 * @date 29 Décembre 2020
 * @warning le fichier "permutation.txt" doit être ua même emplacement que le programme,
sans quoi on obtient une erreur
 * @param byte - caractère/octet à permuter, mis à jour à la suite de la permutation
 */
void permutationWithTableReverse(unsigned char *byte, const unsigned char
*permutationTable){
    int i = 0;
    while (permutationTable[i] != *byte && i < 256) i++;
    *byte = i;
}

/**
 * Multiplication matrice par vecteur
 * La fonction multiplyMatrices() effectue une multiplication entre une matrice et un
vecteur dans cet ordre
 * @date 09 Novembre 2020
 * @warning La matrice doit être de dimension 8x8 et le vecteur de dimension 8x1
 * @param H - matrice de taille 8x8
 * @param v - vecteur V de taille 8x1, mis à jour suite à la multiplication
 */
void multiplyMatrices(unsigned char (*H)[8], unsigned char **v){
    unsigned char vRes[8];
    for (int rowFinal = 0, temp = 0; rowFinal < 8; rowFinal++){
        for (int k = 0; k < 8; k++) temp += H[rowFinal][k] * (*v)[k];
        vRes[rowFinal] = temp % 2; //On fait %2 pour rester en binaire
        temp = 0;
    } for (int i = 0; i < 8; i++) (*v)[i] = vRes[i];
}
```

```c
/**
 * Conversion d'un octet/caractère vers un tableau de bits
 * La fonction byteToBits() prend un octet/caractère et donne en sortie un tableau de
bits correspondant à sa valeur en binaire
 * @date 09 Novembre 2020
 * @note Le bit de poids le plus faible se trouve en position 0
 * @param byte - caractère/octet à convertir
 * @param bits - tableau de bits correspondant à la valeur en binaire de l'octet
 * ### Exemple
 * ~~~~~~~~~~~~~~.c
 *  int byte[8] = byteToBits('c');
 *  for (int i=7; i>=0; i--) printf("%d", byte[i]); //On aura 01100011
 * ~~~~~~~~~~~~~~
 */
void byteToBits(unsigned char *byte, unsigned char **bits){
    *bits = (unsigned char *) calloc(8, sizeof(unsigned char));
    for (int i = 0; (*byte) > 0; i++, *(byte) /= 2) (*bits)[i] = *byte % 2;
}

/**
 * Conversion d'un tableau de bits vers un octet/caractère
 * La fonction bitsToByte() prend un tableau de bits et donne en sortie un
octet/caractère correspondant à sa valeur base 10
 * @date 09 Novembre 2020
 * @param bits - tableau de bits correspondant à la valeur en binaire de l'octet
 * @param byte - octet correspondant à la valeur en base 10 du tableau de bits
 * ### Exemple
 * ~~~~~~~~~~~~~~.c
 *  int bits[] = {1, 1, 0, 0, 0, 1, 1, 0};
 *  printf("%c", bitsToByte(bits); //On aura 'c'
 * ~~~~~~~~~~~~~~
 */
void bitsToByte(const unsigned char *bits, unsigned char *byte){
    for (int i = 0, powerOf2 = 1; i < 8; i++, powerOf2 *= 2) *byte += bits[i] * powerOf2;
}

/**
 * Fonction Matriciel
 * La fonction charApplyMatrix() prend un octet/caractère et va le multiplier par une
matrice H puis effectuer une addition avec C pour encoder cet octet/caractère
 * @date 09 Novembre 2020
 * @note La matrice H et le vecteur C sont connus auparavant
 * @param byte - caractère/octet à encoder, mis à jour suite à l'application de la
fonction
 * @see bitsToByte() byteToBits() multiplyMatrices()
 */
void charApplyMatrix(unsigned char *byte){
    unsigned char *bits = NULL, H[][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
    };
    byteToBits(byte, &bits);
    multiplyMatrices(H, &bits); //Ce qui correspond à H x vi
    unsigned char c[] = {1, 1, 0, 0, 0, 1, 1, 0};
```

```
    for (int j = 0; j < 8; j++) bits[j] = (bits[j] + c[j]) % 2; //Ce qui correspond à Xi
= H x vi + c
    bitsToByte(bits, byte);
    free(bits);
}

/**
 * Fonction Matriciel Inverse
 * La fonction charApplyMatrixReverse() prend un octet/caractère et va le multiplier par
une matrice H' puis effectuer une addition avec C' pour décoder cet octet/caractère après
passage dans charApplyMatrix()
 * @date 09 Novembre 2020
 * @note La matrice H' et le vecteur C' sont connu auparavant
 * @param byte - caractère/octet à décoder, mis à jour suite à l'application de la
fonction
 * @see bitsToByte() byteToBits() multiplyMatrices()
 */
void charApplyMatrixReverse(unsigned char *byte){
    unsigned char *bits = NULL, HPrime[][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
    };
    byteToBits(byte, &bits);
    multiplyMatrices(HPrime, &bits); //Ce qui correspond à H' x Xi
    unsigned char cPrime[] = {1, 0, 1, 0, 0, 0, 0, 0};
    for (int j = 0; j < 8; j++) bits[j] = (bits[j] + cPrime[j]) % 2; //Ce qui correspond
à vi = H' x Xi + c'
    bitsToByte(bits, byte);
    free(bits);
}

/**
 * Fonction XOR logique
 * La fonction applyXOROnByte() prend un octet/caractère et va appliquer l'opération
logique XOR individuellement entre chaque bit de l'octet et la clé d'itération n°2
 * @date 05 Décembre 2020
 * @param byte - caractère/octet à encoder, mis à jour suite à la fonction
 * @param key - 2ème valeur (sous forme d'entier) pour faire le XOR
 * ### Exemple
 * ~~~~~~~~~~~~~~~.c
 *  unsigned char example = 'c'; //Valeur ASCII de 'c': 99
 *  applyXOROnByte(&example, ' '); //Valeur ASCII de ' ': 32
 *  printf("%c", example); //On aura 'C' (écart de 32 entre majuscule et minuscule)
 * ~~~~~~~~~~~~~~~
 * @see bitsToByte() byteToBits()
 */
void applyXOROnByte(unsigned char *byte, const unsigned char *key){
    unsigned char *bits = NULL, *keyBits = NULL, localKey = *key;
    byteToBits(byte, &bits); byteToBits(&localKey, &keyBits);
    for (int i = 0; i < 8; i++) bits[i] = (bits[i] + keyBits[i]) % 2;
    bitsToByte(bits, byte);
    free(keyBits); free(bits);
}

/**
```

```
 * Fonction de Concaténation
 * La fonction concatenate() prend 4 octets/caractères et va les concaténer en respectant
un ordre d'opération précis (voir ci-dessous).
 * Opération effectuées :
 *
 * - 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]
 *
 * Y étant l'entrée et Z la sortie
 * @date 14 Décembre 2020
 * @param byte0 - correspond à Y[0], devient Z[0] suite à l'appel de la fonction
 * @param byte1 - correspond à Y[1], devient Z[1] suite à l'appel de la fonction
 * @param byte2 - correspond à Y[2], devient Z[2] suite à l'appel de la fonction
 * @param byte3 - correspond à Y[3], devient Z[3] suite à l'appel de la fonction
 */
void concatenate(unsigned char *byte0, unsigned char *byte1, unsigned char *byte2,
unsigned char *byte3){
    unsigned char temp0 = *byte0, temp1 = *byte1, temp2 = *byte2, temp3 = *byte3;
    *byte0 = (temp0 + temp1) % 256;
    *byte1 = (temp0 + temp1 + temp2) % 256;
    *byte2 = (temp1 + temp2 + temp3) % 256;
    *byte3 = (temp2 + temp3) % 256;
}

/**
 * Fonction de Concaténation Inverse
 * La fonction concatenateReverse() prend 4 octets/caractères et va inverser la
concaténation de concatenate() en respectant un ordre d'opération précis (voir ci-
dessous).
 * Opération effectuées :
 *
 * - Y[0] = Z[0] - Z[2] + Z[3]
 * - Y[1] = Z[2] - Z[3]
 * - Y[2] = Z[1] - Z[0]
 * - Y[3] = Z[0] - Z[1] + Z[3]
 *
 * Z étant l'entrée et Y la sortie
 * @date 14 Décembre 2020
 * @param byte0 - correspond à Z[0], devient Y[0] suite à l'appel de la fonction
 * @param byte1 - correspond à Z[1], devient Y[1] suite à l'appel de la fonction
 * @param byte2 - correspond à Z[2], devient Y[2] suite à l'appel de la fonction
 * @param byte3 - correspond à Z[3], devient Y[3] suite à l'appel de la fonction
 */
void concatenateReverse(unsigned char *byte0, unsigned char *byte1, unsigned char *byte2,
unsigned char *byte3){
    unsigned char temp0 = *byte0, temp1 = *byte1, temp2 = *byte2, temp3 = *byte3;
    *byte0 = (temp0 - temp2 + temp3) % 256;
    *byte1 = (temp2 - temp3) % 256;
    *byte2 = (temp1 - temp0) % 256;
    *byte3 = (temp0 - temp1 + temp3) % 256;
}

//utiliser "doxygen Doxyfile" pour mettre à jour la documentation
int main(){

printf("================================================================================
==================================\n\n");
    printf("#######  #######  #     #  #######  #######  #######  #######  #######
#######  #######  #     #  #######  #######\n");
```

```
    printf("#         #     #  #     #  #     #     #     #  #         #     #  #
#  #     #  #     #  #         #     #\n");
    printf("#         #######   #    #   #######     #     #     #  #         #######
#######  #######  #######  #         #######\n");
    printf("#         #     #      #     #           #     #     #  #   ###  #     #   #
#  #         #     #  #####    #     #\n");
    printf("#         #     #     #     #           #     #     #  #     #  #     #  #
#  #         #     #  #         #     #\n");
    printf("#######  #     #     #     #           #     #######  #######  #     #  #
#  #         #     #  #######  #     #\n");

printf("\n=================================================================================
==================================\n");
    printf("Bonjour ! Ce programme vous permet d'encrypter/decrypter tous types de
fichier.\n");
    unsigned char action; int N;
    do{
        printf("Que voulez-vous faire ? (0 pour encryptage, 1 pour decryptage)\n");
        scanf(" %c", &action); action -= '0';
    } while (action < 0 || action > 1);
    if (action == 0) printf("Encryptage selectionne, ");
    else printf("Decryptage selectionne, ");

    int size = 0; char *link; unsigned char *fileData = NULL, temp = 1;
    do{
        printf("veuillez entrer le lien du fichier relatif au programme (ex:
../toto.txt)\n");
        link = writeString();
        readFile(link, &fileData, &size);
        if (size < 1){
            fprintf(stderr, ", voulez-vous reessayer ? (0 pour oui, 1 pour non)\n");
            scanf(" %c", &temp); temp -= '0';
            free(link);
            if (temp == 1) return EXIT_SUCCESS;
        }
    } while (size < 1);

    printf("Entrer la cle de chiffrement\n");
    char *encryptionKey = writeString();
    do{
        char temp1[10], *temp2;
        printf("Enfin, entrez le nombre d'iterations a effectuer entre 0 et 999 999
(nombre plus grand, plus de securite mais processus plus long)\n");
        scanf(" %s", temp1);
        if ((N = strtol(temp1, &temp2, 10)) > 999999) N = 999999;
        getchar(); //Elimine le retour à la ligne
    } while (N < 1);

    unsigned char *permutationTable = NULL;
    char *tableLink = NULL;
    readTable("permutation.txt", &permutationTable);
    while (permutationTable == NULL){
        printf("entrer le lien de la table de permutation (ou 0 pour stopper le
programme)\n");
        tableLink = writeString();
        if (tableLink[0] == '0') exit(EXIT_SUCCESS);
        else readTable(tableLink, &permutationTable);
    } free(tableLink);

    int i = 0;
    printProgress(&i, &N);
```

```
    for (i = 1; i <= N; i++){
        unsigned char iterationKey1 = 0, iterationKey2 = 0;
        int keyGeneratorIterator = 0;
        while (encryptionKey[keyGeneratorIterator] != '\0'){ //Création des clés
d'itérations
            int iterationVariation = action == 1 ? N - i + 1 : i;
            iterationKey1 += encryptionKey[keyGeneratorIterator] + iterationVariation;
            iterationKey2 *= encryptionKey[keyGeneratorIterator] + iterationVariation;
            keyGeneratorIterator++;
        }
        if (action == 0){ //Encodage
            for (int j = 0; j < size; j++){
                charPermutation(&fileData[j], &iterationKey1);
                permutationWithTable(&fileData[j], permutationTable);
                charApplyMatrix(&fileData[j]);
                applyXOROnByte(&fileData[j], &iterationKey2);
                if ((j + 1) % 4 == 0) concatenate(&fileData[j - 3], &fileData[j - 2],
&fileData[j - 1], &fileData[j]);
            }
        } else{ //Decodage
            for (int j = 0; j < size; j++){
                if (j % 4 == 0 && j + 3 < size) concatenateReverse(&fileData[j],
&fileData[j + 1], &fileData[j + 2], &fileData[j + 3]);
                applyXOROnByte(&fileData[j], &iterationKey2);
                charApplyMatrixReverse(&fileData[j]);
                permutationWithTableReverse(&fileData[j], permutationTable);
                charPermutationReverse(&fileData[j], &iterationKey1);
            }
        }
        printProgress(&i, &N);
    }
    free(permutationTable); free(encryptionKey);
    writeInFile(fileData, &size, link, &action); //écriture des résultats dans un fichier
    free(link); free(fileData);
    printf("\nAppuyer sur entree pour mettre fin au programme\n"); getchar();
    return EXIT_SUCCESS;
}
```