Université Paris Nanterre

Licence MIASHS première année

Rapport de projet informatique

Chiffrement RSA

Projet réalisé du 10 mars 2022 au 8 mai 2022

Membres du groupe

Badr Agrad
Walid Boudiar

Table des matières

1 Présentation du projet

1.1 Le chiffrement RSA

Nous avons décidé pour notre projet de développer un algorithme de chiffrement utilisant la méthode RSA. Développé par Ronald Rivest, Adi Shamir et Leonard Adleman en 1983, ce procédé informatique nous permet de rendre privé des données (mot de passe, code, texte...) et d'y permettre l'accès uniquement aux détenteurs d'une clé générée aléatoirement. Il existe de nombreuses méthodes de chiffrement, mais l'avantage de la méthode RSA réside dans la complexité des calculs effectués ce qui apporte une grande protection. En effet, le chiffrement RSA est dit asymétrique car il génère une clé privée permettant de chiffrer, et une clé publique permettant de déchiffrer un message. Ainsi, il nous est impossible de déchiffrer sans l'une des deux clés.

1.2 Un peu de mathématiques

Avant toute chose, il est essentiel de comprendre l'ensemble des notions qui seront sollicitées dans l'algorithme RSA. En effet, afin d'être la plus efficace possible, de nombreux outils mathématiques notamment les nombres premiers et l'arithmétique modulaire seront utilisés. En effet, la combinaison de différents calculs permettent à un message de devenir quasiment impossible à décoder

1.2.1 Divisibilité

La divisibilité est un concept élémentaire des mathématiques. Pour rappel, on dit que b divise a,(noté $b \mid a$), si et seulement si on peut exprimer a tel que a = q*b. Logiquement, b ne divise pas a,si a s'écrit sous la forme a = q*b + r, ($r \neq 0$).
Par exemple, 4 divise 12 car la division peut s'écrire 12 = 3*4. Tandis que, 5 ne divise pas 8 car la division s'écrit 8 = 5*1 + 3.
Ainsi, nous pouvons travailler avec le reste de la division entre deux entier que nous appellerons "modulo" noté "%". En effet, cet outil est particulièrement utilisé en informatique comme critère d'arrêt ou encore comme condition.
Voici par exemple un programme qui parcourt un tableau et qui nous permet de savoir si un nombre est pair ou non grâce au reste de la division euclidienne par 2 :

```c
#include <stdio.h>
#define MAX 5

int main(){
    int i;
    int tab[MAX] = {1,2,3,4,5};
    for(i=0; i<MAX; i++){
        if(tab[i]%2 == 0){
            printf("%d est un nombre  pair\n", tab[i]);
        }else{
            printf("%d est un nombre  impair\n", tab[i]);
        }
    }
    return 0;
}
```

(Voici la sortie de l'algorithme) :

```
/tmp/dlwSyDJaau.o
1 est un nombre  impair
2 est un nombre  pair
3 est un nombre  impair
4 est un nombre  pair
5 est un nombre  impair
```

Ainsi, ce critère mathématique nous permet d'étudier et de manipuler les nombres à notre guise. En effet, de nombreuses notions mathématiques sont à l'origine de la divisibilité et de la division euclidienne notamment le plus grand diviseur commun ou encore les nombres premiers...

1.2.2 Les nombres premiers

Un nombre premier est un entier qui n'admet que deux diviseurs distincts : 1 et lui même. Il existe une infinité de nombres premiers et leur répartition est totalement irrégulière.
Par exemple, 5 est premier puisqu'il est divisible par 1 et par 5 (5/1 = 5 et 5/5 = 1), mais aussi car les (ou le) nombres premiers inférieurs à $\sqrt{5}$ ne le divisent pas.
En effet :

- $\sqrt{5} \approx 2.23$
- il existe un unique nombre premier inférieur à $\sqrt{5}$ soit 2
- Or 5 n'est pas divisible par 2 car 5 = 2*4 + 1
- Donc 5 est premier

Ainsi, il existe une liste non exhaustive de nombres premiers allant de 2 à $2^{82589933}$ - 1, soit un nombre long de 24 862 048 chiffres !

Cette propriété est l'une des bases de l'arithmétique et nous permet d'effectuer de nombreuses manipulations. En effet, les nombres premiers permettent de reconstruire les autres nombres, on appelle cela la décomposition en facteurs premiers.
Par exemple, 10 n'est pas un nombre premier, mais 10 = 2*5, où 2 et 5 sont premiers. De même pour 6 = 2*3 ; 9 = 3*3 etc...
En effet, tout nombre strictement supérieur à 1, peut s'écrire en produit de nombres premiers. Cette propriété sera la base de notre algorithme.

1.2.3 L'arithmétique modulaire

Maintenant que nous connaissons la notion de divisibilité et de nombre premiers, nous pouvons entrer dans l'univers de l'arithmétique modulaire. En effet, dans la théorie algébrique des nombres, il existe un ensemble de méthodes qui permet la résolution de problèmes sur les nombres entiers grâce au reste de la division euclidienne. En outre les démonstrations mathématiques et les applications informatiques de ce concept, l'arithmétique modulaire est employée très souvent par le grand public !
Prenons un exemple, imaginez que vous partiez en vacances à 7 heures du matin et que le trajet dure 10 heures. Logiquement, il faut s'attendre à arriver à 5 heures de l'après-midi.
Cela, voudrait donc dire que 7+10 = 5 ?

Pourtant, nous avons appris que 7+10 = 17...
En réalité, il existe deux façons de lire l'heure sur une horloge :

- Soit par rapport à 12 où par exemple, lorsqu'il est 13 heures on dit qu'il est 1 heure de l'après-midi (pm-am chez les anglophones).
- Soit en faisant une petite conversion telle que 1 heure de l'après midi = 13 heures et ainsi de suite...

Pourrait-on donc s'accorder sur le fait que 1 = 13 ?
En fait, l'égalité est partiellement vraie. Sur une horloge, on s'accorde à dire que l'on lit l'heure "par rapport à 12". Ainsi, on dit que ces nombres sont "congrus modulo 12". En effet, en arithmétique modulaire on dit que a est congrus à b modulo n (noté a ≡b[n]) si et seulement si n| b-a. Par récurrence, on admet que si a ≡b[n] alors $a^p \equiv b^p$[n]
Reprenons, l'exemple de notre horloge. On dit que 1≡13[12] car 12-1 = 12 est divisible par 12. De même que 2≡14[12], 4≡16[12], 6≡18[12] etc.
De ce principe de congruence découle le petit théorème de Fermat qui nous sera très utile pour coder et décoder. Le mathématicien français expliquait que pour un nombre premier p et un entier a non divisible par p alors p| a^{p-1} - 1.
En effet, soit :

- a^{p-1} - 1 = kp ,k∈ Z
- a^{p-1} = 1 + kp
- $a^{p-1} \equiv 1$[p]
- Par récurrence, $(a^{p-1})^n \equiv 1^n$[p]
- a^{pn-n} - 1 ≡ 0[p]
- a^{pn-n} - 1 = pk ,k ∈ Z

C'est donc ce théorème que nous allons utiliser pour coder et décoder des messages.
Dorénavant, vous avez toutes les notions requises afin de comprendre le système RSA.

1.3 Le fonctionnement de l'algorithme

Le chiffrement RSA est, tel qu'il est mentionné ci-dessus, asymétrique : cela sollicite la création d'une clé publique pour chiffrer et d'une clé privée pour déchiffrer. En effet, la clé publique est connu des expéditeurs et permet de coder les messages qu'ils souhaitent transmettre à leurs destinataires, qui eux décodent.
Avec la méthode RSA si Alice veut envoyer un message chiffré à Bob, elle chiffre le message avec la clé publique puis envoie à Bob le texte chiffré. Bob peut ensuite déchiffrer le texte avec sa propre clé privée.

1.3.1 La clé privée et la clé publique

Voici maintenant comment générer mathématiquement la clé publique et la clé privée :

- Il faut choisir deux nombres premiers distincts (assez grands) p et q tel que le pgcd(p,q) = 1 et p et q distincts.
- Soit n, le produit de ces deux nombres tel que n = p*q
- Il faut calculer $\phi(n)$ = (p-1)*(q-1)
- On pose un entier e tel que 0 < e < $\phi(n)$ et e est premier avec $\phi(n)$ (pgcd(e,$\phi(n)$) = 1)
- Ensuite on pose d est égale à l'inverse de e modulo $\phi(n)$ tel que d*e ≡ 1[$\phi(n)$]

Ainsi notre clé publique sera le couple (n,e) ou (e,n). Tandis qu'on utilisera d en tant que clé privée.

1.3.2 Chiffrer et déchiffrer

Maintenant que nous sommes en possession de nos clés, il est temps de chiffrer un message. Le chiffrement RSA fonctionne en transformant un caractère en un nombre (long de plusieurs chiffres).

- Soit un caractère représenté par l'entier M (par exemple : la position du caractère dans un alphabet/tableau des ASCII etc...) et M < n
- On élève cet entier à la puissance e soit M^e
- On pose C entier qui représentera notre message chiffré
- Ainsi, l'entier C doit remplir cette condition C $\equiv M^e$[n]

Une fois que le message est chiffré, voici comment le déchiffrer à l'aider de notre clé privée :

- On élève l'entier C à la puissance n soit C^e
- Dorénavant, l'entier M, soit le caractère déchiffré, est calculé tel que M$\equiv C^d$[n]
- De cette manière la nous obtenons notre position initiale dans l'alphabet

Voici un petit exemple simple d'utilisation :

- On prend deux nombres entiers distincts soit p = 3, q = 11
- On calcule n = 3*11 = 33
- On calcule $\phi(n)$ = (3-1)*(11-1) = 2*10 = 20
- On pose e = 7 (car e < $\phi(n)$ et pgcd($\phi(n)$,e) = 1)
- On cherche d tel que 7d $\equiv$ 1[20], cela nous donne d = 3. En effet 3*7 = 21$\equiv$1[20]
- Soit la lettre C qui correspond à la troisième de l'alphabet. On calcule 3^7[33] = 2187[33] grâce à la division euclidienne on obtient 2183[33] = 9
- Enfin pour décoder, on calcule 9^3[33] = 729[33] = 3. On retombe bien sur 3, correspondant à la lettre

Bien évidemment les algorithmes, utilisant le RSA, génèrent de très grands nombres premiers de manière à rendre le chiffrement incassable.
Dorénavant, vous avez toutes les notions requises pour comprendre et appliquer la méthode RSA !

2 Le programme informatique

2.1 En quoi consiste-t-il ?

Nous avons donc décidé de construire un algorithme qui génère les clés publiques et privées et qui chiffre et déchiffre un message déjà existant. En effet, cela nous permettra de manipuler des fichiers texte à notre guise. Voici la liste de ce que notre code doit savoir faire pour chiffrer et déchiffrer :

— Générer de grands nombres premiers
— Comparer les nombres premiers (premiers entre eux, plus grand diviseur communs entre deux nombres etc...)
— Être capable de lire un caractère dans un fichier.txt et de le manipuler
— Créer un fichier.txt et écrire dedans

2.2 Les outils utilisés

2.2.1 Système d'exploitation et langage

Notre programme sera codé en C sur le logiciel Geany dans le système d'exploitation Linux. En effet, la compilation (procédé par lequel l'ordinateur va transformer un texte en une suite d'instructions binaires) est d'autant plus accessible sur Linux que sur Windows, la commande "gcc" (permettant la compilation) étant facilement utilisable. De plus, Geany permet de compiler à l'aide d'un simple bouton.

2.2.2 Bibliothèques

Le C est un langage dit "de bas niveau". Contrairement à d'autres langages comme *Python*, où la machine facilite certaines tâches et offre certaines fonctionnalités, en C nous partons de zéro et devons tout définir nous même. Ainsi, inclure des bibliothèques nous permettra de coder de manière plus efficace. Voici donc la liste des bibliothèques qui seront utilisés dans ce programme :

— "stdio.h" bilbiothèque de base en C
— "stdlib.h" bibliothèque de base qui offre des fonctions plus poussées notamment 'malloc' et 'calloc' qui nous serons utiles
— "stdbool.h" bibliothèque qui permet de manipuler les opérateurs booléens (true, false)
— "string.h" bibliothèque qui permet de manipuler les chaines de caractères
— "time.h" bibliothèque qui permet de manipuler les informations temporelles
— "unistd.h" bibliothèque que nous allons utiliser pour faire des pauses dans l'algorithme.

Nous développerons davantage l'intérêt d'utiliser chaque bibliothèque dans le descriptif de l'algorithme.

3 Le cahier des charges

3.1 Les besoins et problématiques

Notre algorithme RSA doit être capable de générer aléatoirement une clé publique et privée, de chiffrer et déchiffrer un message, pouvoir l'afficher dans la console, le tout en offrant une interface simple d'utilisation.
L'algorithme RSA opère d'une certaine manière :
Dans un premier temps, la machine va lire un caractère pour ensuite identifier sa position dans l'alphabet. Logiquement, les opérations seront opérées par rapport à cette position représentée par un entier. Le résultat des calculs du chiffrement sera donc un entier qui, lui aussi, est censé représenter une position. L'objectif étant, lors de l'opération de décryptage, de retomber sur l'entier initial.
Cependant, il faut comprendre le fonctionnement du traitement de caractère dans le langage C.

3.1.1 La table des ASCII

Supposons qu'un ordinateur fonctionne en fonction de l'alphabet anglophone. Ainsi, selon cet alphabet la lettre 'A' représente la première lettre (donc la position 1), la lettre 'C' la troisième et ainsi de suite jusqu'à 26. Cependant, qu'en est-il pour le caractère '@' ou encore '♯' ? De même que lorsque vous définissez un mot de passe, comment l'ordinateur pourrait faire la distinction entre une majuscule et une minuscule là où l'alphabet classique n'en fait pas ?
En réalité, l'ordinateur traite l'information d'une manière différente. En fait, le type "char" (caractère) est manipulé comme un entier (soit la position de ce char). Il existe un tableau, déjà implémenté dans la bibliothèque stdio.h, listant un ensemble de caractères de 0 à 127 incluant l'alphabet majuscule, minuscule, les espaces, tirets, parenthèses et de nombreux autres symboles.
Voici la Table des ASCII listant l'ensemble des caractères pouvant être lus par un ordinateur :

ASCII TABLE

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]

Par exemple, la lettre 'a' est le 97e caractère de la table des ascii, tandis que 'A' le 65e. Voici une exécution simple permettant de constater le lien entre entiers et ca-

ractères.

```
printf("%c = %d\n", 'a', 'a');
printf("%c = %d\n", 'A','A');
```

Sortie :

```
a = 97
A = 65
```

La table reste cependant limitée par le fait qu'elle ne comporte pas de lettres "spéciales" comme le 'ç' ou encore le 'é'.
De ce fait, il nous sera plus pratique de créer notre propre alphabet, complet, avec un maximum de caractères. (Ce sera un tableau représentant une chaîne de caractère utilisé en tant que constante)

```
#define ALPHABET "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz !:;,.?/_-'+-#(){}[]<>\n\t0123456789éèàçùêî"
```

Nous pourrons donc manipuler tout type de caractères.

3.2 Les solutions

Nous avons donc établi une stratégie permettant à l'algorithme d'opérer de la meilleure des manières le chiffrement RSA. Voici, chaque étape numérotée une à une :

1. Générer deux nombres premiers tels que le produit de ces deux nombres soit au moins supérieur à 5000 (plus pratique pour calculer et plus sécurisé)
2. Calculer la clé publique et la clé privée et les communiquer à l'utilisateur
3. Ouvrir un fichier "message.txt" déjà existant
4. Créer un fichier "crypt.txt"
5. Lire "message.txt" caractère par caractère
6. Trouver la position de chaque caractère selon l'alphabet que l'on a défini
7. Chiffrer les différents caractères et écrire les blocs de nombres obtenus dans "crypt.txt"
8. Afficher le message crypté dans la console
9. Fermer les deux fichiers
10. Ouvrir le fichier "crypt.txt" et créer un fichier "decrypt.txt"
11. Lire "crypt.txt" bloc par bloc
12. Déchiffrer chaque bloc et écrire chaque caractère obtenu dans "decrypt.txt"
13. Afficher le message décrypté dans la console
14. Fermer les deux fichiers
15. Quitter le programme

3.3 Le résultat attendu

Ainsi, le programme doit être capable de générer (n,e) et (d) sans aucune aide de l'utilisateur. Pour ceci, nous allons définir une fonction qui calcule deux nombres premiers distincts à partir de la méthode du crible d'eratosthène. Ensuite, nous pourrons en calculer le produit, définir ϕ, puis e à partir d'une génération aléatoire et d'une fonction qui détermine si deux nombres sont premiers entre eux (pgcd(e,ϕ) = 1 et e<ϕ). Enfin, il nous sera capable d'établir d grâce à une fonction calculant l'inverse modulaire. Voici un exemple de ce que l'algorithme devra calculer :

```
les deux nombres premiers sont 127 et 193
Voici la clé publique : (24511,5)
Voici votre clé privée : 9677
```

Ainsi, pour un message donné, le programme doit être capable de le chiffrer sous cette forme :

```
[15275][16541][20091][09599][13911][02338][09696]
[16541][23719][00303][17553][15976][15616][18052]
[09317][16541][23091][21554][09599][13911][13322]
[23663][23663][23663]
```

Puis, de le déchiffrer afin de retrouver le message initial :

```
Vive l'informatique !
```

4 Le Code

4.1 Le typage

La spécificité du chiffrement RSA réside dans le fait qu'il faut manipuler de grands nombres premiers. En effet, si les algorithmes manipulaient de petits nombres premiers (comme par exemple 3 et 11), il serait facile de "casser le RSA" car le message chiffré ne serait pas assez complexe. En effet, de grands nombres premiers rendent le RSA quasiment infaillible à moins de disposer d'un ordinateur quantique, pouvant effectuer des millions de calcul à la fois...
En C, nous disposons du type entier classique "int" qui occupe une certaine taille d'octets en mémoire. Nous utilisons une machine virtuelle sur un ordinateur à 64 bits. Grâce à l'opérateur "sizeof(int)", nous avons pu observer que le type "int" occupe seulement 4 octets en mémoire. De plus, la valeur maximale que le type "int" peut contenir est 32 767.
Par conséquent, cela limite notre algorithme de seulement utiliser des nombres premiers inférieurs à 150. Au-delà, la machine manipule des nombres qu'un entier ne peut pas contenir. Je vous laisse deviner la réaction de l'ordinateur [1]...
Nous allons donc manipuler le type "long" qui peut contenir des entiers allant de -2 147 483 647 à 2 147 483 647. Sur notre machine de 64-bits, le type "long" occupe 8 octets soit deux fois plus que le type int.

```c
void main(){
    printf("le type int occupe : %d d'octets !\n", sizeof(int));
    printf("le type long occupe : %d d'octets !\n", sizeof(long));
}
```

```
le type int occupe : 4 d'octets !
le type long occupe : 8 d'octets !
```

Nous allons aussi créer un type qui sera notre clé (publique et privée inclues dedans). Ce type sera une structure contenant trois longs : n,e et d. Cela nous permettra de manipuler les clés plus facilement.

```c
typedef struct cle{
    long n;
    long e;
    long d;
}key;
```

1. Saviez-vous qu'en 1996, cette erreur a causé l'explosion de la fusée Ariane 5 lors du vol 501 ? Pour des raisons financières, l'algorithme de la fusée Ariane 4 a été simplement copié dans la fusée Ariane 5. Cependant, les capteurs de la fusée envoyaient des nombres flottants de 64 bits tandis que l'ancien algorithme ne pouvait manipuler que des entiers de 16 bits. Ainsi, après réception d'un flottant qui ne pouvait pas être contenu par la mémoire de l'algorithme, la fusée, pensant aller dans une mauvaise direction, a explosé. Elle aura tenu 35 secondes.

4.2 Fonctions

Notre algorithme fonctionne pas à pas exactement tel qu'il est énoncé dans le cahier des charges. Le code est découpé en dix-sept fonctions. En voici la liste :

- **int pgcd(long a, long b**)
  Cette fonction classique permet de calculer le plus grand diviseur commun entre deux nombres. Elle sera récursive.
- **int premier_entreeux(long a, long b)**
  A l'aide de la fonction pgcd, nous pouvons calculer si deux entiers sont premiers entre eux. Cela est vrai si et seulement si,ils admettent tout deux 1 comme plus grand diviseur commun.
- **long chercher_cara(char c)**
  Cette fonction nous permet de trouver l'emplacement d'un caractère dans l'alphabet que nous avons défini au départ. Si la lettre est dans l'alphabet on retourne sa position, sinon -1.
- **long puissance(long n, int p)**
  Cette fonction nous permet de calculer un long n^p. Elle sera récursive.
- **char creer_nombre(long val, long taille)**
  Cette fonction va nous permettre de créer un bloc de nombre de la forme "00...c", avec c un entier correspondant à l'emplacement dans l'alphabet d'un caractère. L'autre paramètre de la fonction est la taille du bloc qui sera égale aux nombres de chiffres de (n). Ainsi, la fonction initialise un pointeur de caractères à NULL, un compteur à 0 et une variable max qui reçoit $10^{taille-1}$. Le pointeur de caractères va nous permettre d'allouer dynamiquement un espace mémoire pour des char, taille fois (soit un tableau de char).Par exemple, si un char est compris dans 1 octet et que taille = 5, alors on alloue un espace de 5 octets. Ainsi, on définit une boucle qui s'arrête lorsque max est nul où, à l'aide de la division euclidienne et de son reste, on rempli le tableau un certain nombres de fois par 0, puis par l'entier val de sorte à ce qu'il soit entièrement compris dans le tableau.On retourne ce tableau. La fonction peut être affichée uniquement grâce à l'opérateur "%s". Voici un exemple d'utilisation de la fonction

  ```
  printf("%s", creeNombre(15,5));
  ```
  et sa sortie : `00015`

- **long crypter_lettre(long c, key a)**
  Les caractères sont considérés selon le long trouvé dans chercher_cara)
  Cette fonction s'appuie sur le Petit théorème de Fermat. On réalise e occurrences où, à chaque tour de boucle, on multiplie l'entier c (soit le caractère chiffré initialisé à 1) par le caractère initial et on divise par n le résultat trouvé de manière à ne garder que le reste de la division. (On utilisera l'opérateur %). La fonction retourne c.
- **long decrypter_lettre(long c, key a)**
  La fonction opère exactement de la même manière que pour crypter une lettre. Cependant, plutôt que de faire e occurrences, elle en fera d. Ceci afin de bien respecter l'algorithme RSA. La fonction retourne c.

— **long crible_eratosthène(long val1, long val2, long *p)**
Le crible d'eratosthène est un algorithme mathématique permettant de trouver les nombres premiers inférieurs ou égaux à un entier. Soit une liste croissante de nombres entiers débutant à 2. Selon cette liste il faut barrer tous les multiples de 2, sauf 2. On fait de même pour tous les multiples de 3 et on continue jusqu'à la racine carrée du plus grand entier de notre liste.
Dans notre code, nous allons rajouter une spécificité à cet algorithme. En effet, voulant générer des nombres premiers au moins supérieurs à 300, nous avons décider de réaliser le crible d'érathostène entre un intervalle (un minimum et un maximum). L'allocation dynamique nous a permis de facilement réaliser cette fonction. Nous aurons besoin d'un tableau de booléens initialisé à 0 par la fonction calloc (false) et d'un tableau de long qui contiendra les nombres premiers. Grâce à deux boucles imbriquées, nous pourrons obtenir la liste des nombres premiers qui sera stockée dans le tableaux d'entiers.
— **void generer_premier(long *p1, long *p2)**
Ainsi, grâce au crible d'eratosthène et à la fonction rand, nous pouvons générer deux nombres premiers distincts. Pour des raisons de performances, nous choisirons des nombres compris entre 300 et 1000.
— **long inverse_modulo(long a, long m)**
Cette fonction, nous sera utile au calcul de la clé privée. Ainsi, selon l'algorithme d'Euclide étendu, on cherche i tel que a*i $\equiv 1[\phi]$. Donc pour i allant de 1 à m, s'il existe un i tel que le reste de a/m * le reste de i/m le tout modulo m est égal à 1, alors on retourne i.
— **void calcul_key(key *a)**
Cette fonction va nous permettre, à l'aide de toutes celles que l'on a défini, de calculer nos deux clés. Elle prend en paramètre un pointeur de structure. Ainsi, en passant en paramètre l'adresse d'une structure initialisée dans notre main, la clé recevra les valeurs calculées par la fonction. Le pointeur reçoit à la fin, pour chaque variable de la structure, les valeurs des entiers n, e et d.
— **long taille_bloc(long n)**
Pour définir la taille d'un bloc, la fonction retourne le nombre de chiffres de n.
— **void crypter_fichier(char *nom, key a)**
La fonction prend en paramètre un pointeur sur un caractère (soit une chaîne de caractère de la forme abcd[] = "abdc") et la structure key. On ouvre deux fichiers, un déjà existant et un autre que nous allons créer nommé "crypt.txt". Ainsi, la fonction va récupérer chaque caractère du texte et va trouver sa position dans l'alphabet. Selon cette position, on crypte le caractère puis on l'écrit sous forme de bloc dans le deuxième fichier grâce à la fonction creeNombre.
On obtient donc un bloc de nombres que l'on peut afficher dans la console.
— **void decrypter_fichier(key a, long taille, char ALPHABET[])**
La fonction prend en paramètre une structure clé, la taille de l'alphabet et l'alphabet (chaine de caractère). Après l'ouverture des fichiers "crypt.txt" et "decrypt.txt", nous devons récupérer chaque bloc de nombres. Pour ce faire, nous devons créer une boucle while nous permettant de parcourir chaque caractère jusqu'à la fin du fichier. Ensuite, une variable "val" nous permettra de stocker chaque bloc de nombres et de le décrypter à l'aide de la fonction. Enfin, le reste de la division entre val et la taille de l'alphabet nous permettra d'avoir la position exacte de la lettre décryptée dans l'alphabet. Nous pourrons donc l'écrire dans notre fichier "decrypt" et dans la console.

— **void affichage()**
— **void interactif(char *c)**
Ces deux fonctions serviront pour l'affichage dans la console. On utilisera la fonctionnalité pause de la bibliothèque "unistd.h", qui nous permettra d'offrir un affichage plus agréable.

4.3 Fonctions n'ayant pas pu être réalisées

— Casser le chiffrement RSA.
Au départ, nous voulions trouver une manière de récupérer un message chiffré par la méthode RSA. Cependant, la solution fut trouvée assez tard dans l'avancement de notre projet. En effet, nous pensions faire une analyse de fréquence du message chiffré, puis nous ne savions pas quelles autres étapes il fallait réaliser. Cette méthode, est utilisé pour casser un chiffrement de césar, par exemple. Cependant, le RSA est asymétrique. Donc, cela est abordable plus "facilement". Il faut partir du fait que l'on possède la clé publique. A partir de n, il faudra réaliser une décomposition en facteurs premiers. Si l'on trouve p et q, alors nous pouvons réaliser l'ensemble des calculs nécessaires pour trouver d et déchiffrer aisément.
Néanmoins, cela est beaucoup trop long. En effet, pour des grandes valeurs de n, la décomposition pourrait prendre des siècles. Il faudrait se munir d'un ordinateur quantique malheureusement pas encore commercialisé...
— Une interface graphique Nous souhaitions réaliser une interface graphique à l'aide de la bibliothèque "sdl.h". N'étant pas présente sur Geany, il fallait l'installer. Cependant, l'installation de sdl a échoué de nombreuses fois sur la machine virtuelle sur laquelle nous avons travaillé. La bibliothèque et toutes ses fonctionnalités furent bien téléchargés mais l'appel des fonctions de base de sdl comme "SDL CreateWindow" ne fonctionnait pas et générait des erreurs. Nous avons donc opté pour un affichage dans la console.

5 Répartition du travail

Badr Agrad : réalisé les fonctions permettant de calculer la clé publique et privée, de cryptage et décryptage (lettre, creernombre, fichier), de générations de nombres premiers et le rapport.
Walid Boudiar : réalisé des fonctions d'arithmétique (pgcd, premiers entreeux), la fonction pour chercher les caractère etles deux fonctions d'affichage.

6 Difficultés rencontrées

6.1 Difficultés rencontrées par Badr Agrad

”Ma première difficulté était de maîtriser les pointeurs et la mémoire. En effet, j'ai passé beaucoup de temps à me documenter un maximum que cela soit sur youtube, openclassroom ou encore des livres sur le C afin de comprendre au mieux le fonctionnement de la mémoire. De même pour les pointeurs de structures et la création de type, où beaucoup d'erreurs étaient faites au départ ce qui rendait le code impossible à exécuter. Il y a aussi eu le problème de compilation de la bibliothèque sdl qui l'a rendu inutilisable. Enfin, le fait de réaliser un algorithme de chiffrement RSA en C, en prenant en compte certaines problématiques que l'on a pu rencontrer, était assez compliqué”

6.2 Difficultés rencontrées par Walid Boudiar

”A titre personnel, j'ai déjà des difficultés avec le langage C (notamment les notions de pointeurs, structures). J'ai donc du réaliser des fonctions à ma portée. Néanmoins, je n'ai pas eu de problème à comprendre l'algorithme de chiffrement RSA qui, mathématiquement, est moins dense qu'il n'en a l'air.”

7 Bilan

7.1 Le Bilan de Badr Agrad

”Ce projet m'a permis de passer un cap en programmation. En effet, j'étais impatient d'acquérir le plus de connaissances possible pour enfin coder quelque chose de réellement utile. J'ai énormément progressé en C. J'ai pu apprendre de nouvelles méthodes pour coder, mieux comprendre la mémoire mais aussi comment la compilation fonctionnait. En effet, j'ai pu observer à quel point ce langage est essentiel car il permet de comprendre le fonctionnement d'un ordinateur de la meilleure des manières. J'ai aussi progressé en arithmétique. Ce projet m'a permis de confirmer que je voulais travailler en tant qu'ingénieur développeur ! ”

7.2 Le Bilan de Walid Boudiar

”Le projet m'a aidé à surmonter les difficultés que j'avais en informatique. J'ai aimé le fait de pouvoir travailler en binôme, de se partager des tâches, des idées. Au départ, j'étais réticent vis à vis du chiffrement RSA mais finalement le projet fut très intéressant.”

7.3 Conclusion et perspectives

En conclusion, nous avons tous deux aimé participer à ce projet. En effet, ce fut très enrichissant de coder un algorithme pouvant être utilisé par n'importe qui. Le chiffrement RSA est énormément utilisé aujourd'hui que cela soit pour chiffrer vos mots de passe ou encore informations bancaires. Étant tous deux passionnés par la cyber-sécurité, comprendre et appliquer un algorithme de cryptographie si utile dans la protection de données fut une expérience enrichissante. De plus, les contraintes énoncées par le corps enseignant ont rendu ce projet comme un véritable défi à relever.
Nous comptons continuer à enrichir notre algorithme durant les prochaines années, afin de développer le code le plus efficace et performant possible. L'apprentissage du web développement, nous permettra même de réaliser un site ou une application qui sollicitera notre code.

8 Sources

[1] Sebastien Varrette et Nicolas Bernard, Programmation avancée en C, Hermes Science publication, 2007

[2] Algorithme RSA, Wikipédia

[3] Cours d'harmonisation, Claire Hanen, François Metayer, 2021

[4] Tutoriel langage C, de "formation vidéo", Youtube

9 Annexes

Annexe A : Exemple d'exécution du code

Le chiffrement RSA étant un outil, nous avons donc décidé de rendre notre projet le plus ludique possible. Connaissez vous le manga One piece ? C'est le récit d'un jeune garçon prenant la mer et voulant devenir le Roi des pirates. Pour atteindre son but, il devra trouver le légendaire "One piece" trésor laissé par Gold Roger le Seigneur des mers exécuté par le Gouvernement Mondial. Le rapport avec le chiffrement RSA ? Dans l'histoire de One Piece, il existe ce qu'on appelle des ponéglyphes qui sont des stèles cubiques sur lesquels sont gravés des messages par les habitants du "siècle oublié". C'est un époque, volontairement censuré par le gouvernement, qui regorge de mystères. Cependant, il faut maîtriser le langage de l'ancien monde pour déchiffrer ces stèles. C'est là que nous intervenons ! En effet, l'algorithme RSA nous a permis de déchiffrer les ponéglyphes ! Voici ce que l'algorithme vous demandera :

```
Etes vous bien en quête du Légendaire One Piece ? [o/n] :
```

Voici les clés :

```
Voici la clé publique : (21583,3)
Voici votre clé privée : 14187
```

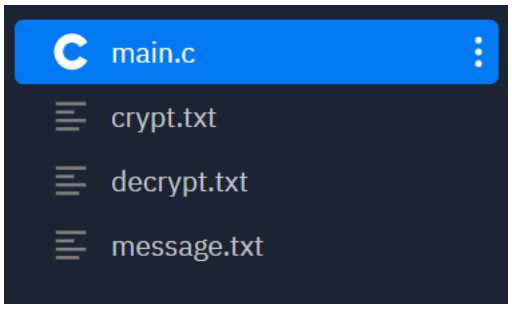
Voici maintenant à quoi ressemble le message lorsqu'il est déchiffré :

```
VOICI A QUOI RESSEMBLE LE PONEGLYPHE DANS L'ANCIEN LANGAGE :
[30089][55292][73068][24537][16618][74516][29391][22062][16618][55919][11065][67297][61042][11065][67297][34820][11065][67297][16618][38909]
[17443][46362][55292][29391][16618][79545][38800][61042][31139][22062][74516][75840][17443][29391][16618][17443][24537][34820][16618][38909]
[29391][16618][74516][29391][64867][79545][24537][55919][74516][29391][16618][10100][46362][55292][16618][64867][34820][38800][42437][24537]
[74516][61042][22062][16618][10100][46362][00092][55292][79545][16618][53316][55292][61042][55292][34820][24537][55292][22062][16618][17443]
[34820][16618][79545][74516][29391][16618][79545][55292][18339][11065][67297][34820][74516][34820][16618][38909][74516][16618][79545][74516]
[34820][29391][16618][29391][38800][46362][53316][53316][34820][24537][61042][64867][74516][29391][28520][16618][26018][61042][16618][55292]
[61042][38800][34820][74516][16618][64867][74516][17443][74516][61042][38909][24537][61042][22062][16618][29391][55292][16618][30089][55292]
[24537][16618][24537][16618][34820][11065][67297][74516][79545][79545][74516][75840][74516][61042][22062][16618][74516][43638][55292][29391]
[11065][67297][16618][38800][46362][16618][61042][38800][61042][28520][27900][27900][35572][67535][38800][00092][29391][31100][35572][67535]
[08475][16618][46362][61042][16618][24537][61042][64867][55292][74516][61042][16618][75840][74516][75840][18339][34820][74516][16618][38909]
[16618][00032][76967][46370][16618][10100][46362][55292][16618][24537][16618][38909][11065][67297][61760][11065][02201][16618][11065][67297]
[11065][67297][16618][74516][75840][17443][34820][55292][29391][38800][61042][61042][11065][67297][08475][16618][24537][16618][74516][61042]
[74516][61042][38909][46362][16618][17443][24537][34820][79545][74516][34820][16618][38909][74516][16618][79545][24537][16618][79545][11065]
[31139][74516][61042][38909][74516][16618][17443][24537][34820][16618][46362][61042][16618][31139][24537][34820][38909][55292][74516][61042]
[38909][74516][16618][17443][34820][55292][29391][38800][61042][16618][74516][22062][16618][24537][16618][74516][61042][29391][46362][55292]
[74516][16618][64867][38800][75840][75840][74516][61042][64867][11065][67297][16618][11065][02201][16618][17443][34820][55292][74516][34820]
[30089][55292][73068][24537][16618][38909][24537][61042][29391][16618][79545][74516][16618][18339][46362][22062][16618][38909][00092][24537]
[22062][11065][67297][61042][46362][74516][34820][16618][29391][74516][29391][16618][17443][34820][38800][17443][34820][74516][29391][16618]
[38800][46362][53316][53316][34820][24537][61042][64867][74516][29391][28520][16618][75750][38800][34820][29391][10100][46362][74516][16618]
[67535][38800][00092][29391][31100][35572][67535][38800][16618][24537][16618][34820][74516][75840][24537][34820][10100][46362][11065][67297]
[10100][46362][74516][16618][79545][74516][16618][75840][11065][83388][75840][74516][16618][31139][24537][34820][38909][55292][74516][61042]
[38909][74516][16618][17443][34820][55292][29391][38800][61042][16618][10100][46362][55292][16618][79545][46362][55292][16618][24537][55919]
[55292][22062][16618][17443][24537][34820][79545][11065][67297][16618][38909][74516][16618][79545][24537][16618][79545][11065][67297][31139]
[61042][38909][74516][16618][24537][55919][24537][55292][22062][16618][29391][38800][46362][38909][24537][55292][61042][74516][75840][74516]
```

Enfin, le message déchiffré :

```
EN VOICI LA SIGNIFICATION :
Fortune, Gloire et pouvoir, cet homme avait amasse toutes les richesses du monde, son nom Gol-D-Roger. Ces dernières paroles incitèrent les hommes
de toute la planète à s' aventurer en mer...
 - Mon trésor, je vous le laisse si vous voulez , trouvez le je l'ai laissé quelque part dans ce monde...-
Tous se lancèrent sur la route de Grandline , dans l'espoir de mettre la main sur ce fameux trésor. Le monde entier connu alors une grande vague de
 piraterie...En espérant que cela vous aidera dans votre périple...
```

Dans le dossier contenant le code, il n'y aura au départ qu'un seul fichier texte. Après exécution il y en aura trois, avec le fichier "crypt.txt" qui se modifiera en fonction des clés générées. En effet, le fichier décrypté contiendra exactement le même contenu que le fichier initial (auquel cas, le chiffrement aura échoué).



Annexe B : Manuel utilisateur

Pour utiliser le code, il faudra posséder tous les fichiers "projet test.c", "projet test.o", "projet test" et "One Piece.txt". Il faut ouvrir l'invite de commande, parcourir le dossier contenant les fichiers puis exécuter "projet test"!

Annexe C : Bonus

Nous avons tout de même réalisé un fichier à partir des fonctions du code, permettant de générer des nombres premiers compris entre 1000 et 10 000 000 ! Nous avons aussi défini une fonction qui permet de tester si le nombre est bien premier (au cas où il y aurait une erreur). Nous avons du utiliser des types long long afin que les opérations soient possibles. Au delà de 10 000 000, l'algorithme à malheureusement du mal à générer les nombres premiers... Voici un exemple d'exécution de l'algorithme.