**Projet M3101 :**

**Sujet : Cassage d’empreint MD5**

**Réalisé par : CASTEL Jérémy et DI-NARDO Valentin**

**Sommaire**

[Introduction : 3](#__RefHeading___Toc1456_3614413215)

[Comment utiliser : 3](#__RefHeading___Toc1458_3614413215)

[Ordinateur de test : 4](#__RefHeading___Toc1460_3614413215)

[Utilisation RAM : 4](#__RefHeading___Toc1462_3614413215)

[Algorithme séquentiel : 5](#__RefHeading___Toc1464_3614413215)

[Algorigramme séquentiel : 7](#__RefHeading___Toc1466_3614413215)

[Algorigramme de la fonction Main 7](#__RefHeading___Toc1468_3614413215)

[Algorigramme de la fonction incrémente 8](#__RefHeading___Toc1470_3614413215)

[Algorigramme de la fonction reset 9](#__RefHeading___Toc1472_3614413215)

[Temps Enregistrés : 10](#__RefHeading___Toc1474_3614413215)

[Diagrammes : 11](#__RefHeading___Toc1476_3614413215)

[Explications : 15](#__RefHeading___Toc1478_3614413215)

[Au niveau des temps : 15](#__RefHeading___Toc1480_3614413215)

[Avantages : 18](#__RefHeading___Toc1482_3614413215)

[Désavantages : 18](#__RefHeading___Toc1484_3614413215)

[Pourquoi le 26 threads ? : 19](#__RefHeading___Toc1486_3614413215)

# Introduction :

En utilisant la librairie crypto++, nous devons trouver la chaîne de caractères en claire d'une empreinte MD5. Pour cela nous allons tester toutes les combinaisons possibles jusqu'à obtenir la bonne chaîne MD5.

Lien docs : <https://www.cryptopp.com/wiki/Md5>

Lien GitHub : https://github.com/Jeremy-JRM/M3101-MD5-BruteForce

# Comment utiliser :

Lancer un terminal dans le dossier où se trouve le script « compil.sh » puis lancer le script avec la commande « ./compil.sh ». Cela va d’abord compiler la librairie Crypto++ que l’on utilise pour force brute puis faire des tests avec différentes longueurs de mots et différents nombres de threads et enfin vous générer plusieurs fichiers. Le script peut durer plus de 5 min donc soyez patients.

Le fichier « temps.log » vous montre les résultats des tests effectués par le script.

Le fichier « thread disposition.log » vous montre comment sont disposer nos threads.

Dans le dossier « src » vous allez retrouver tous le code.

Dans le dossier « lib », la lib Crypto++.

Dans le dossier « info » toutes les informations utiles concernant les performances de votre PC.

Dans le dossier « build » tous les exécutables que vous pouvez aussi exécuter à la main avec la commande :

- ./multiThread.exe <nombre de thread(2,4,8,26)> <mot à trouver>

- sequentiel.exe <mot a trouver>

Dans le dossier « algo » tous nos algorithmes et algorigrammes expliquant notre code.

# Ordinateur de test :

* **Nombre de CPU :** 8
* **Nombre de cœurs :** 4
* **Fréquence :**  4,00 GHz
* **Cache :** L1 = 256 KiB

L2 = 1 MiB

L3 = 8 MiB

* **RAM :** 8 Go

# Utilisation RAM :

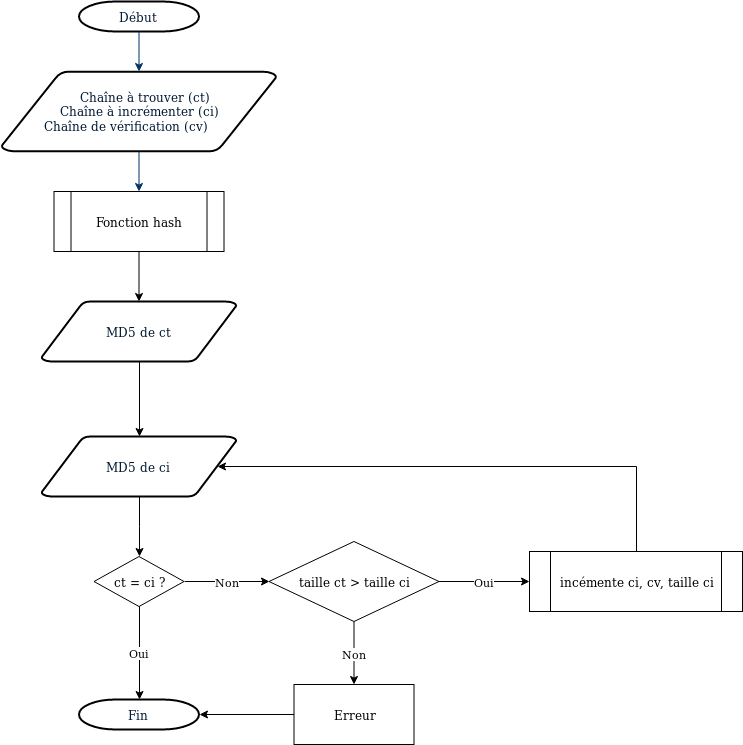
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Séquentiel | 2 Threads | 4 Threads | 8 Threads | 26 Threads |
| RAM avant le lancement | 1,14 G | 1,14 G | 1,14G | 1,14 G | 1,14 G |
| RAM pendant le lancement | 1,15 G | 1,15 G | 1,15 G | 1,15 G | 1,15 G |

# Algorithme séquentiel :

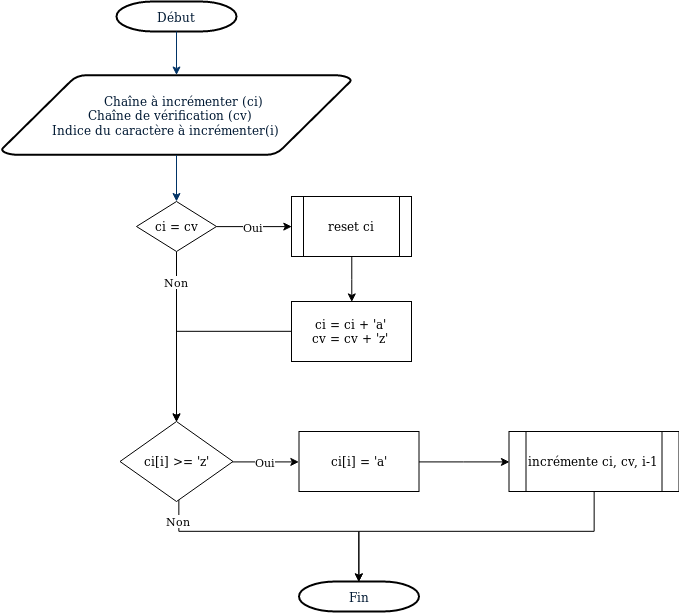
|  |
| --- |
| Variables Globales : r : Chaîne de caractères  t : Booléen  nt : Entier  Fonction md5 (→*chaine* : Chaîne de caractères) : Chaîne de caractères  Donnée(s) : *chaine* (en entrée) Le mot à transformer  Résultat : Retourne la valeur md5 du mot à transformer  Procédure reset (↔*s* : Chaîne de caractères)  Donnée(s) : *s* (en entrée/sortie) Le mot à remettre à zéro  *l* (en entrée) La lettre à mettre au début du mot  Résultat : Remet le mot en entrée à zéro avec une certaine lettre  Variable locale : *i* : Entier  Début  *s*[0] ← l  Pour *i* de 1 à taille(*s*) Faire  *s*[i] ← 'a'  FinPour  Fin  Procédure inc (↔*s* : Chaîne de caractères, ↔*v* : Chaîne de caractères, →*i* : Entier, →*l* : Caractère)  Donnée(s) : *s* (en entrée/sortie) Le mot à incrémenter  *v* (en entrée/sortie) Le mot de vérification de fin  *i* (en entrée) La position de la lettre à incrémenter  *l* (en entrée) La lettre de départ du mot incrémenter  Résultat : Incrémente le mot s pour tester la prochaine combinaison de lettre  Début  Si (*i* < 0 OU *i* > taille(*s*)) Alors  Ecrire "ERREUR : indice impossible"  Fin  FinSi  Si (*s* = *v*) Alors  reset(*s*, l)  *s* ← *s* + 'a'  *v* ← *v* + 'z'  Sinon  Si (*i* = 0) Alors  *s*[*i*]← *s*[*i*] + (*nT* - 1)  FinSi  Si (*s*[*i*] ≥ 'z') Alors  *s*[*i*] ← 'a'  inc(*s*, *v*, *i* - 1, *l*)  Sinon  *s*[*i*] ← *s*[*i*] + 1;  FinSi  FinSi  Fin  Procédure dechiffre (→*s* : Chaîne de caractères, →*v* : Chaîne de caractères)  Donnée(s) : *s* (en entrée) Le mot de combinaisons  *v* (en entrée) Le mot de vérification de fin  Résultat : Test le mot de combinaisons avec le mot à trouver jusqu'à ce qu'ils soient égaux  Variables Locales : *l* : Caractères  *dS* : Chaîne de caractères  *dR* : Chaîne de caractères  Début  *l* ← *s*[0]  *dS* ← md5(*s*)  *dR* ← md5(*r*)  TantQue (*dS* ≠ *dR* OU *t* ≠ Vrai) Faire  inc(*s*, *v*, taille(*s*) - 1, l)  *dS* ← md5(*s*)  FinTantQue  Si (*dS* = *dR*) Alors  *t* ← Vrai  FinSi  Fin |

# Algorigramme séquentiel :

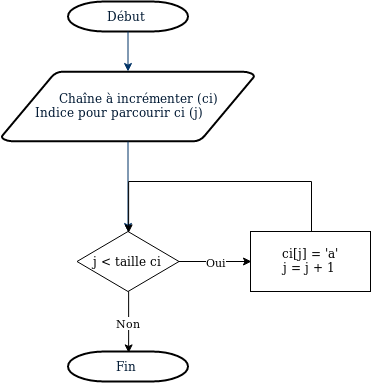
## Algorigramme de la fonction Main



## Algorigramme de la fonction incrémente

****

## Algorigramme de la fonction reset

****

# Temps Enregistrés :

**Mot: za**

**=========**

Nombre Thread(s): 1 | Temps: 0.001 seconds

Nombre Thread(s): 2 | Temps: 0 seconds

Nombre Thread(s): 4 | Temps: 0 seconds

Nombre Thread(s): 8 | Temps: 0 seconds

Nombre Thread(s): 26 | Temps: 0.02 seconds

**Mot: zza**

**=========**

Nombre Thread(s): 1 | Temps: 0.033 seconds

Nombre Thread(s): 2 | Temps: 0.017 seconds

Nombre Thread(s): 4 | Temps: 0.009 seconds

Nombre Thread(s): 8 | Temps: 0.009 seconds

Nombre Thread(s): 26 | Temps: 0.098 seconds

**Mot: zzza**

**=========**

Nombre Thread(s): 1 | Temps: 0.945 seconds

Nombre Thread(s): 2 | Temps: 0.465 seconds

Nombre Thread(s): 4 | Temps: 0.31 seconds

Nombre Thread(s): 8 | Temps: 0.32 seconds

Nombre Thread(s): 26 | Temps: 0.178 seconds

**Mot: zzzza**

**=========**

Nombre Thread(s): 1 | Temps: 21.815 seconds

Nombre Thread(s): 2 | Temps: 13.428 seconds

Nombre Thread(s): 4 | Temps: 10.013 seconds

Nombre Thread(s): 8 | Temps: 10.116 seconds

Nombre Thread(s): 26 | Temps: 8.649 seconds

**Mot: zzzzza**

**=========**

Nombre Thread(s): 1 | Temps: 569.078 seconds

Nombre Thread(s): 2 | Temps: 352.668 seconds

Nombre Thread(s): 4 | Temps: 261.679 seconds

Nombre Thread(s): 8 | Temps: 259.085 seconds

Nombre Thread(s): 26 | Temps: 209.579 seconds

# Diagrammes :

# Explications :

## Au niveau des temps :

Nos threads sont arrangés de sortes à ce qu’ils reçoivent une lettre de départ et une lettre de fin et qu’ils parcourent les premières lettres de leurs mots en fonction du nombre de threads utilisés, par exemple :

Pour 2 threads :

- 1er thread : lettre de départ : a

lettre de fin : y

lettres parcourues : a, c, e, g, i, k, m, o, q, s, u, w, y

- 2eme thread : lettre de départ : b

lettre de fin : z

lettres parcourues : b, d, f, h, j, l, n, p, r, t, v, x, z

- Parcours de l’alphabet par les threads : Toutes les 2 lettres

Pour 4 threads :

- 1er thread : lettre de départ : a

lettre de fin : y

lettres parcourues : a, e, i, m, q, u, y

- 2eme thread : lettre de départ : b

lettre de fin : z

lettres parcourues : b, f, j, n, r, v, z

- 3eme thread : lettre de départ : c

lettre de fin : w

lettres parcourues : c, g, k, o, s, w

- 4eme thread : lettre de départ : d

lettre de fin : x

lettres parcourues : d, h, l, p, t, x

- Parcours de l’alphabet par les threads : Toutes les 4 lettres

Pour 8 threads :

- 1er thread : lettre de départ : a

lettre de fin : y

lettres parcourues : a, i, q, y

- 2eme thread : lettre de départ : b

lettre de fin : z

lettres parcourues : b, j, r, z

- 3eme thread : lettre de départ : c

lettre de fin : s

lettres parcourues : c, o, s

- 4eme thread : lettre de départ : d

lettre de fin : t

lettres parcourues : d, l, t

- 5eme thread : lettre de départ : e

lettre de fin : y

lettres parcourues : e, m, u

- 6eme thread : lettre de départ : f

lettre de fin : v

lettres parcourues : f, n, v

- 7eme thread : lettre de départ : g

lettre de fin : w

lettres parcourues : g, o, w

- 8eme thread : lettre de départ : h

lettre de fin : x

lettres parcourues : h, p, x

- Parcours de l’alphabet par les threads : Toutes les 8 lettres

Les mots vont être parcourus de cette façon (voir « thread disposition.txt » pour plus de précisions) :

Pour 2 threads :

1er thread 2eme thread

1ere 2eme Etc 1ere 2eme Etc

aa ca ba da

ab cb bb db

ac cc bc dc

ad cd bd dd

ae ce be de

af cf bf df

ag cg bg dg

ah ch bh dh

ai ci bi di

aj ck bj dj

ak ck bk dk

al cl bl dl

am cm bm dm

an cn bn dn

ao co bo do

ap cp bp dp

aq cq bq dq

ar cr br dr

as cs bs ds

at ct bt dt

au cu bu du

av cv bv dv

aw cw bw dw

ax cx bx dx

ay cy by dy

az cz bz dz

Et ainsi de suite avec toutes les lettres parcourues de chaque thread

Les itérations se font donc de la forme :

Pour 2 lettres : P + A

Pour 3 lettres : P + A + A

Pour 4 lettres : P + A + A + A

Etc ..

P = Lettres Parcourues/ A = Lettres de l’alphabet

# Avantages :

L’avantage de faire fonctionner nos threads de cette façon et que le parcours se fera dans l’ordre de l’alphabet donc les mots commençant par les lettres du début de l’alphabet seront plus rapidement trouvés, le nombre de threads augmente cela car les threads vont couvrir un plus grand espace de mots dès la 1ere itération de chaque thread.

Exemple :

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1er thread | 2eme thread | 3eme thread | 4eme thread | 5eme thread | 6eme thread | 7eme thread | 8eme thread |
| 2 thread | a | b |  |  |  |  |  |  |
| 4 thread | a | b | c | d |  |  |  |  |
| 8 thread | a | b | c | d | e | f | g | h |

Lettres parcourues dès la 1ere itérations de chaque threads

Donc cela montre que pour un mot commençant par ‘a’ le 2, 4 ,8 thread auront à peu près le même temps, pour un mot commençant par ‘c’ alors ce sera le 4 et 8 threads et enfin pour un mot commençant par ‘f’ ce sera le 8 threads.

# Désavantages :

Le désavantage est donc simple, les mots commençant par les dernières lettres de l’alphabet sont donc plus longs à trouver que les mots en début d’alphabet quelques soit le nombre de thread. Les 8 threads seront quand même le plus rapide dans ce cas la.

Pour remédier à cela nous avons décidé d’utiliser les 26 threads.

# Pourquoi le 26 threads ? :

C’est simple, 26 lettres de l’alphabet donc 26 threads, leur lettre début sera donc leur lettre de fin et ils ne parcourront que les mots qui ont comme première lettre leur lettre attribuée.

L’avantage de faire cela est que quelle que soit la première lettre du mot que l’on cherche le temps sera le même dans tous les cas, les 2, 4, 8 threads seront toujours plus rapides pour les premières lettres mais pour les autres lettres (comme le montre nos diagrammes avec des mots commençant par la lettre ‘z’) les 26 threads seront lui le plus rapide. En moyenne les 26 threads seront les plus intéressants sachant que la première lettre du mot n’est pas connue.

Ici on remarque que pour 26 threads le temps est à peu près le même et que cela devient intéressant à partir de ‘n’ par rapport aux 2 threads et a partir de ‘x’ par rapport au 4 et 8 threads. Ce résultat et encore plus marqué lorsque le nombre de lettres du mot est augmenté (les tests seraient un peut long pour avoir les temps avec 6 lettres).

On peut donc conclure que les 26 threads sont utiles quand le mot commence au moins par la lettre ‘q’ et qu’il a au moins une taille de 5 lettres.