Université du Québec à Rimouski

Campus de Lévis

RAPPORT

TRAVAIL PRATIQUE #2

Par

Bastien Goulet et Cédrick Larrivée

Travail présenté à M. Martin Arsenault.

Dans le cadre du cours Sécurité informatique.

INF36207-MS

8 mars 2023

Table des matières

[Algorithme MD5 3](#_Toc128245293)

[Application dictionnaire 3](#_Toc128245294)

[Application hachage 3](#_Toc128245295)

[Solution 3](#_Toc128245296)

[Résultats 3](#_Toc128245297)

[Conclusion 3](#_Toc128245298)

[Annexes 3](#_Toc128245299)

[Références bibliographiques 3](#_Toc128245300)

# Algorithme MD5

Afin d’utiliser un hash cryptographique selon l’algorithme MD5 dans notre application, nous avons fait appel à la librairie *System.Security.Cryptography* du framework .NET Standard. En effet, cette librairie offre un choix important de différentes implémentations d’algorithmes connus afin d’effectuer l’encryption, la décryption ou encore de calculer les hash selon des algorithmes comme le MD5, le SHA-1 ou encore le SHA-512.

La classe que nous avons utilisée a été la classe *MD5*, tout simplement. Celle-ci, étant une classe abstraite, ne peut pas être instanciée directement. Il faut plutôt faire appel à une méthode *Create* qui instancie, elle, la classe par défaut qui s’occupe de l’algorithme MD5. Une fois cette étape complétée, il reste à traiter la chaîne de caractères dont nous voulons calculer le hash. Peu importe la chaîne de caractères passée à la fonction à sens unique offerte par notre objet MD5, celle-ci nous retournera nécessairement une chaîne de caractères contenue sur 128 bit. Il est à noter que cette chaîne est toujours identique à condition que la chaîne en entrée est la même également. Malheureusement, l’algorithme MD5 ne répond pas parfaitement au requis des fonctions de hash selon lequel deux entrées différentes ne devraient pas produire le même hash. En effet, avec l’algorithme MD5, il est possible d’obtenir des collisions de hash relativement facilement.

Malgré ce défaut, cet algorithme fait parfaitement l’affaire dans le cadre de ce travail. Donc, pour l’utiliser, il suffit de passer un tableau d’octets (notre chaîne de caractère en entrée transformée) à la méthode *ComputeHash* de l’objet MD5 et celui-ci effectue les calculs nécessaires qui sont assez complexes, d’ailleurs. Pour convertir notre entrée en tableau d’octets utilisable, nous avons fait appel à la librairie *System.Text.Encoding* du framework .NET Standard afin de normaliser notre chaîne de caractères en Unicode et de la convertir en tableau d’octets. Nous débutons donc par appeler la méthode *Normalize* sur la chaîne en entrée et nous passons le résultat à la méthode *GetBytes* de la classe *UTF-8* de la librairie *Encoding* afin d’extraire les octets de notre chaîne après l’avoir converti de l’UTF-16 à l’UTF-8. D’ailleurs, dans cet encodage, la plupart des caractères sont contenus sur un octet.

La méthode *ComputeHash* fait ensuite son travail et calcule le hash qui nous est retourné, lui aussi, dans un tableau d’octets. La dernière étape du processus consiste à construire une chaîne de caractères qui contient chaque octet du hash MD5 sous sa représentation hexadécimale. Pour y parvenir, nous utilisons donc une *StringBuilder* et nous y ajoutons, un après l’autre, le caractère hexadécimal en faisant appel à la méthode *ToString(“X2”)* sur chaque octet.

# Application dictionnaire

L’application dictionnaire est relativement simple d’utilisation. Il suffit d’entrer dans les bons champs la longueur minimale et maximale des mots de passe générés. Par la suite, l’utilisateur doit sélectionner les différents caractères qui seront utilisés dans la génération des mots de passe. Pour ce faire, il suffit de cocher les options qui vous intéressent. Notez que vous pouvez aussi entrer des caractères au choix dans le champ à cet effet. Vous devrez ensuite sélectionner un emplacement pour sauvegarder votre fichier ainsi que le nommer. Finalement, il ne vous reste plus qu’à cliquer sur le bouton de génération. Le dictionnaire des mots de passe sera généré à l’endroit indiqué précédemment. Évidemment, si l’utilisateur ne remplit pas une de ces conditions, un message d’erreur apparaitra à l’écran.

# Application hachage

L’application de hachage est elle aussi très simple d’utilisation. Premièrement, l’utilisateur doit sélectionner un dictionnaire de mots de passe en format texte dans la première fenêtre. Une fois la sélection du dictionnaire complétée, une autre fenêtre s’ouvre. Il est possible de voir, en haut à droite de la fenêtre, le nombre de tentatives ainsi que le temps écoulé depuis le début de la lecture du dictionnaire. Au centre de la fenêtre se trouve une zone pour entrer le hash qui dont la décryption sera tentée. Enfin, on retrouve, en bas à droite, le nombre de mots total dans le dictionnaire. Ce nombre total correspond, en réalité, au nombre de lignes lues dans le dictionnaire au moment du chargement du fichier texte. Le dictionnaire fourni par notre professeur étant formaté de cette manière, nous avons présumé qu’une ligne du fichier, peu importe le dictionnaire, correspondait à un mot de passe unique.

Une fois la tentative de découverte du mot de passe lancée, le programme lit les mots de passe du dictionnaire, ligne par ligne, en appelant la méthode *ComputeHash* du service *Md5Hasher* afin de calculer le hash *MD5* du mot de passe en question. Chaque hash ainsi calculé est comparé avec celui fourni par l’utilisateur. Si deux valeurs sont identiques, l’application a découvert le mot de passe correspondant au hash inscrit. Le cas échéant, cette information est communiquée à l’utilisateur. Sinon, le hash ne possède pas une équivalence dans le dictionnaire.

## Solution

|  |  |
| --- | --- |
| 937557fac5cffc250ccf72031474078b | G2Ma1N |
| 2414766fb5121dfebdf220d4b8a550a0 | Ma1s0N |
| 973d2d342378f637aeeb9ec96f5a4b46 | B0nB0n |
| 6e63c6bf94b66e04ff2f48be546f0110 | ababbababb |
| 90b2360c704e0a805cee6dd6fd71eeaa | zyxyzxyxxz |
| 154e5cae63af27012426896ab9da2ac0 | divergances |
| 139230ff1ef21bb767bb9b475cf99873 | donnnaient |
| 7746fcc0b507887866f8bd227b37122c | Donetsks |
| db50a77debe1141e236bfefb468440ab | Eau-de-viiie |

# Résultats

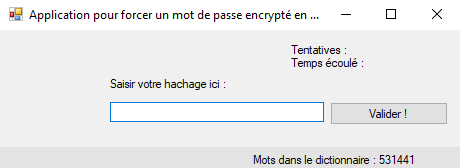
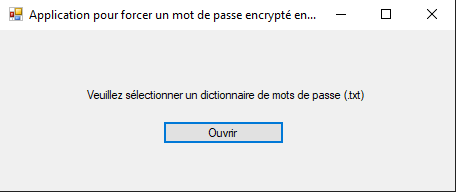
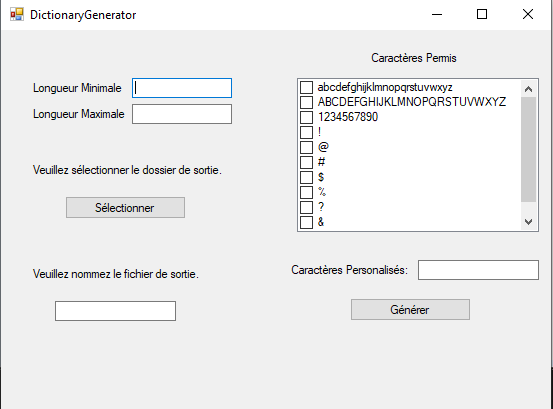
L’utilisateur du site csharpexamples.com, turgay, fut d’une grande aide pour nous afin de mettre en place un mécanisme de hachage selon l’algorithme MD5. En effet, les classes et les méthodes nécessaires de la librairie *System.Security.Cryptography* afin d’y parvenir nous ont été indiquées par cet utilisateur. La référence vers la page en question est accessible dans la section des références en annexe à ce rapport.

L’utilisation de l’architecture en couches n’était pas un bon choix sans en être un mauvais. Par contre, cela reste une bonne pratique, car cette architecture permet d’avoir un projet qui est bien structuré. Nous avons rencontré un problème lors de la génération des mots de passe pour le dictionnaire. Une approche avec fonction récursive a été utilisée. Cela causait des problèmes de débordement de mémoire quand l’application devait générer des mots à 5 et plus. Après recherche, nous avons trouvé une solution. L’utilisation de yeild lors du retour dans des fonctions de type IEnumerable permet de pallier à ce problème, car il permet de donner la prochaine valeur d’une itération quand il est utilisé.

# Conclusion

En conclusion, nos applications répondent bien aux divers critères qui étaient exigés. Notre application dictionnaire permet la création de dictionnaires volumineux. Aucun problème n’a été soulevé pour générer un dictionnaire de plus 56 gigaoctets. En ce qui concerne notre application de hachage, elle aussi fonctionne bien en plus d’être plus simple d’utilisation que notre application de génération de dictionnaire.

# Annexes



# Références bibliographiques

Acamar. “Openfiledialog Does Not Close.” *Openfiledialog Does Not Close*, 20 Nov. 2013, https://social.msdn.microsoft.com/Forums/vstudio/en-US/e32fd547-c9ac-4287-a0a8-cb777fe67da3/openfiledialog-does-not-close?forum=vbgeneral.

Adegeo. “Make Thread-Safe Calls to Controls - Windows Forms .NET Framework.” *Make Thread-Safe Calls to Controls - Windows Forms .NET Framework | Microsoft Learn*, https://learn.microsoft.com/en-us/dotnet/desktop/winforms/controls/how-to-make-thread-safe-calls-to-windows-forms-controls?view=netframeworkdesktop-4.8.

BillWagner, and pkulikov. “Yield Statement - Provide the next Element in an Iterator.” *In an Iterator | Microsoft Learn*, 12 Feb. 2022, https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/statements/yield.

Dotnet-Bot. “Filedialog Class (System.windows.forms).” *(System.Windows.Forms) | Microsoft Learn*, https://learn.microsoft.com/en-us/dotnet/api/system.windows.forms.filedialog?redirectedfrom=MSDN&view=windowsdesktop-7.0.

Turgay. C# Create A MD5 Hash From A String | *csharpexamples.com*, https://csharpexamples.com/c-create-md5-hash-string/