

Sécurité des Systèmes

TP 1 : Hachage

Réalisé par les étudiantes : JOUIJATE Rim et AGOULZI Imane

Filière : Sécurité des Systèmes d'Information (SSI)

Année universitaire : 2022-2023

Dans ce TP, nous commençons par quelques recherches bibliographiques à propos de Hachage, ensuite nous passons à la réalisation d'une interface qui permet de réaliser des opérations de Hachage.

1 – Généralités sur le Hachage :

Fonction de Hachage :

Une fonction de hachage est un algorithme qui peut être exécuté sur des données, comme un fichier individuel ou un mot de passe, pour produire une valeur appelée checksum.

L'utilisation principale de cet algorithme est de vérifier l'authenticité d'une donnée. En effet, deux fichiers ne peuvent être considérés identiques que si les checksums générés en utilisant la même fonction de hachage sur chaque fichier sont identiques.

Les caractéristiques de la fonction de hachage :

Elle est unidirectionnelle et irréversible.

Sa sortie sera toujours de la même longueur indépendamment de la taille d'entrée.

De chaque entrée, elle produit une sortie unique.

Exemples de fonctions de hachage :

- **MD5** (message-digest algorithm): c'est un algorithme de hachage développé en 1991 par Ronald L. Rivest et il a été spécifié en 1992 comme RFC 1321. Cet algorithme prend en entrée n'importe quelle taille de données numériques et produit une sortie de 128 bits. Cet algorithme est généralement utilisé, d'une part pour authentifier les messages ainsi que la vérification du contenu et les signatures numériques, et d'autre part pour vérifier l'intégrité des données contre la corruption involontaire.
- **SHA-1** (Secure Hash Algorithm 1) : c'est un algorithme de hachage développé par la National Security Agency (NSA) en 1995, il prend une entrée et produit une valeur de hachage de 160 bits. Cette dernière est connue comme un message digest qui est généralement rendu comme un nombre hexadécimal de longueur 40 chiffres. Mais malheureusement, il a été cassé théoriquement en 2005 et pratiquement en 2017.

- **SHA-2** (Secure Hash Algorithm 2) : une famille d'algorithmes de hachage de sécurité développés par l'Agence nationale de la sécurité des systèmes d'information (NSA) pour remplacer les algorithmes de la famille SHA-1 qui ont été considérés comme étant insuffisamment sécurisés. Cette famille elle comporte les fonctions, SHA-256 et SHA-512 dont les algorithmes sont similaires mais la taille de mot est différente, en effet, la taille de mot est de 32 bits pour SHA-256 et 64 bits pour SHA-512. Pour SHA-224 et SHA-384 qui sont essentiellement des versions des précédentes dont la sortie est tronquée (cela veut dire que la longueur de la sortie est réduite ou qu'on a modifié certains de ses éléments), et plus récemment SHA-512/256 et SHA-512/224 qui sont des versions tronquées de SHA-512. Le dernier suffixe indique le nombre de bits du hash.
- **SHA-3** (Secure Hash Algorithm 3) : SHA-3 utilise un algorithme de hachage fondamentalement différent de celui de SHA-2, appelé permutation-based hash. Ce type d'algorithme a été choisi pour remplacer les algorithmes basés sur compression utilisés dans SHA-1 et SHA-2 en raison de sa robustesse accrue face aux attaques. Pour le moment, cet algorithme n'est pas cassé.
- **BLAKE2b et BLAKE2s** sont des algorithmes de hachage. Ils sont conçus pour être plus rapides et plus sécurisés que les algorithmes de hachage classiques tels que SHA-256 et SHA-3. BLAKE2b est optimisé pour les processeurs 64 bits, il produit des hashes 256 bits. Tandis que BLAKE2s est optimisé pour les processeurs 32 bits et les appareils mobiles et est plus rapide que SHA-256 sur ces plateformes, il produit des hachages de 160 bits. Les deux algorithmes sont considérés comme sécurisés et sont souvent utilisés dans les protocoles de sécurité pour vérifier l'intégrité des données transmises.

2 – Réalisation de l'interface :

Nous avons décidé de réaliser l'interface en utilisant le langage de programmation Python, from scratch.

Cette interface nous permet de générer le hash d'une chaîne de caractères données en entrée ainsi de le comparer avec un autre hash donné par l'utilisateur ou bien généré à partir de cette interface.

Dans la partie gauche, nous avons ajouté une liste déroulante contenant différentes fonctions de hachage (de la bibliothèque hashlib de Python) afin que l'utilisateur puisse choisir la fonction qu'il veut travailler avec, puis il doit cliquer sur « Valider » pour valider son choix.

The screenshot shows the 'HACHAGE' application window. On the left, under the title 'HACHAGE', there is a section 'Fonction de Hashage :'. It features a dropdown menu with the text 'Choisissez !' and a list of hash functions: SHA1, SHA224, SHA256, SHA384, SHA512, BLAKE2b, BLAKE2s, and MD5. Below the list is a button labeled 'Valider mon choix'. To the right of this section are two main input areas. The top area is labeled 'Message :' and has a text input field and a 'Générer' button. Below it is a 'Hash :' label with an empty text input field. The bottom area is labeled 'Message à vérifier :' and also has a text input field and a 'Générer' button. Below this is a 'Hash (*) :' label with an empty text input field. At the very bottom of the window are three buttons: 'Copier', 'Vérifier', and 'Terminer'.

Dans la partie droite en haut, l'utilisateur peut saisir le message qui veut générer son hash en cliquant sur le bouton « générer ».

This screenshot shows the same 'HACHAGE' application window, but now the 'Message :' field contains the text 'Salut à tous'. The 'Hash :' field now displays the generated hash: '8bcd6307add2a2042ee7bb479c5ee95e0e88f8'. A message 'Hash généré avec succès !' is visible to the right of the hash field. The dropdown menu on the left now shows 'SHA1' as the selected function. The 'Message à vérifier :' section remains empty. The 'Copier', 'Vérifier', and 'Terminer' buttons are still at the bottom.

Ainsi qu'en bas, c'est exactement la même chose.

De plus le bouton copier permet de copier le message qui se trouve dans le champ « Message » dans le champ « Message à vérifier ».

En ajoutant le bouton « Vérifier » qui permet de comparer si les deux hashes s'ils sont identiques ou non.

The screenshot shows the HACHAGE application window. On the left, the title 'HACHAGE' is displayed above a 'Fonction de Hashage :' dropdown menu set to 'SHA1'. A 'Valider mon choix' button is at the bottom of this section. The main area has two sections. The top section, 'Message :', contains a text input with 'Salut à tous' and a 'Générer' button. To the right, it says 'Hash généré avec succès !'. Below this, the 'Hash :' field displays '8bcd6f6307add2a2042ee7bb479c5ee95e0e88f8'. The bottom section, 'Message à vérifier :', also has a text input with 'Salut à tous' and a 'Générer' button. To the right, it says 'Les deux Hashs sont identiques !'. Below this, the 'Hash (*) :' field displays the same hash: '8bcd6f6307add2a2042ee7bb479c5ee95e0e88f8'. At the very bottom, there are three buttons: 'Copier', 'Vérifier', and 'Terminer'.

This screenshot shows the HACHAGE application with a different verification result. The 'Message :' and 'Message à vérifier :' sections both contain the text 'Salut à tous !'. The 'Hash :' field still shows '8bcd6f6307add2a2042ee7bb479c5ee95e0e88f8'. However, the 'Hash (*) :' field now displays a different hash: '8cbf9d14a13afe839be824924d0a7cc7d91f4efd'. The status message on the right now reads 'Les deux Hashs ne sont pas identiques !'. The 'Copier', 'Vérifier', and 'Terminer' buttons remain at the bottom.

Et finalement, le bouton « Terminer » sert à fermer l'interface.

Références :

<https://www.avast.com/c-md5-hashing-algorithm>

<https://www.lifewire.com/what-is-sha-1-2626011>

<https://www.geeksforgeeks.org/sha-1-hash-in-java/>

<https://infosecwriteups.com/breaking-down-sha-3-algorithm-70fe25e125b6>

https://csrc.nist.gov/CSRC/media/Events/ISPA-DECEMBER-2013-MEETING/documents/new_sha3_functions.pdf

<https://www.comparitech.com/blog/information-security/what-is-sha-2-how-does-it-work/>

<https://www.techopedia.com/definition/30571/secure-hash-algorithm-2-sha-2>

<https://www.blake2.net/>

[https://en.bitcoinwiki.org/wiki/BLAKE_\(hash_function\)](https://en.bitcoinwiki.org/wiki/BLAKE_(hash_function))

Annexe :

Notre interface codée en Python :

```
interface.py > ...
1  import customtkinter
2  from tkinter import *
3  import hashlib
4
5  customtkinter.set_appearance_mode("dark") # Modes: "System" (standard), "Dark", "Light"
6  customtkinter.set_default_color_theme("blue") # Themes: "blue" (standard), "green", "dark-blue"
7
8  app = customtkinter.CTk()
9  app.geometry("900x600")
10 app.title("Hachage")
11
12 frame = customtkinter.CTkFrame(master=app, width=900, height=600)
13 frame.pack(pady=20, padx=60, expand=True)
14
15 frame_1 = customtkinter.CTkFrame(master=frame, height=430)
16 frame_1.place(x=20, y=30)
17
18 frame_21 = customtkinter.CTkFrame(master=frame, width= 510)
19 frame_21.place(x=250, y=30)
20
21 frame_22 = customtkinter.CTkFrame(master=frame, width= 510)
22 frame_22.place(x=250, y=260)
23
24 #####functions declaration#####
25
26 def close_window():
27     app.quit()
28
29 def hash2(b):
30     hash2_value.set(b)
31
32 def hash1(b):
33     hash1_value.set(b)
34
```

```
interface.py > ...
35 def get_value_choix():
36     print(choix_function.get())
37
38 def input_msg_1():
39     user_input = msg_1.get("1.0", "end").strip()
40     print(user_input)
41
42 def input_msg_2():
43     user_input = msg_2.get("1.0", "end").strip()
44     print(user_input)
45
46 def copier() :
47     msg_2.delete("1.0", "end")
48     msg_2.insert("1.0", msg_1.get("1.0", "end").strip())
49
```

```

50 def hashing1():
51     f = choix_function.get()
52     user_input = msg_1.get("1.0", "end").strip()
53
54     if (f == 'SHA1') :
55         result = hashlib.sha1(user_input.encode())
56     if (f == 'SHA224') :
57         result = hashlib.sha224(user_input.encode())
58     if (f == 'SHA256') :
59         result = hashlib.sha256(user_input.encode())
60     if (f == 'SHA384') :
61         result = hashlib.sha384(user_input.encode())
62     if (f == 'SHA512') :
63         result = hashlib.sha512(user_input.encode())
64     if (f == 'BLAKE2b') :
65         result = hashlib.blake2b(user_input.encode())
66     if (f == 'BLAKE2s') :
67         result = hashlib.blake2s(user_input.encode())
68     if (f == 'MD5') :
69         result = hashlib.md5(user_input.encode())
70
71     hash1(result.hexdigest())
72
73     rep_1 = customtkinter.CTkLabel(master=frame_21, justify=customtkinter.LEFT, text="Hash généré \navec succès !")
74     rep_1.place(x=405, y=100)
75
76 def hashing2():
77     f = choix_function.get()
78     user_input = msg_2.get("1.0", "end").strip()
79
80     if (f == 'SHA1') :
81         result = hashlib.sha1(user_input.encode())
82     if (f == 'SHA224') :
83         result = hashlib.sha224(user_input.encode())
84     if (f == 'SHA256') :
85         result = hashlib.sha256(user_input.encode())
86     if (f == 'SHA384') :
87         result = hashlib.sha384(user_input.encode())
88     if (f == 'SHA512') :
89         result = hashlib.sha512(user_input.encode())
90     if (f == 'BLAKE2b') :
91         result = hashlib.blake2b(user_input.encode())
92     if (f == 'BLAKE2s') :
93         result = hashlib.blake2s(user_input.encode())
94     if (f == 'MD5') :
95         result = hashlib.md5(user_input.encode())
96
97     hash2(result.hexdigest())
98
99 def verify() :
100     if len(hash1_value.get())!=0 and len(hash2_value.get())!=0 :
101         if (hash1_value.get() == hash2_value.get()) :
102             rep_2 = customtkinter.CTkLabel(master=frame_22, justify=customtkinter.LEFT, text="Les deux Hashs \nsont identiques !")
103         else :
104             rep_2 = customtkinter.CTkLabel(master=frame_22, justify=customtkinter.LEFT, text="Les deux Hashs \nne sont pas \nidentiques !")
105         rep_2.place(x=405, y=100)
106
107     ##### frame 1 #####
108
109     hashage = customtkinter.CTkLabel(master=frame_1, justify=customtkinter.LEFT, text="HACHAGE", font=
110     ("Helvetica", 36, "bold"))
111     hashage.place(x=10, y=10)
112
113     choix = customtkinter.CTkLabel(master=frame_1, justify=customtkinter.LEFT, text="Fonction de Hashage :")
114     choix.place(x=10, y=70)
115
116     choix_function = customtkinter.StringVar()
117     hash_function = customtkinter.CTkOptionMenu(frame_1, values=["SHA1", "SHA224", "SHA256", "SHA384",
118     "SHA512", "BLAKE2b", "BLAKE2s", "MD5"], variable=choix_function)
119     hash_function.place(x=10, y=100)
120     hash_function.set("Choisissez !")
121
122     valider_choix = customtkinter.CTkButton(master=frame_1, text="Valider mon choix",
123     command=get_value_choix)
124     valider_choix.place(x=10, y=350)

```

```

123 ##### frame 21 #####
124
125 message_1 = customtkinter.CTkLabel(master=frame_21, justify=customtkinter.LEFT, text="Message :")
126 message_1.place(x=10, y=10)
127
128

```

```

interface.py > hashing2
129 msg1 = customtkinter.StringVar()
130 msg_1 = customtkinter.CTkTextbox(master=frame_21, width=350, height=70)
131 msg_1.place(x=10, y=40)
132
133
134
135 hashage_1 = customtkinter.CTkLabel(master=frame_21, justify=customtkinter.LEFT, text="Hash :")
136 hashage_1.place(x=10, y=120)
137
138
139 hash1_value = customtkinter.StringVar()
140 hash_1 = customtkinter.CTkEntry(master=frame_21, width=350, textvariable=hash1_value)
141 hash_1.place(x=10, y=150)
142
143
144 button_generer1 = customtkinter.CTkButton(master=frame_21, text="Générer", width=100, command=hashing1)
145 button_generer1.place(x=400, y=10)
146
147
148
149
150 ##### frame 22 #####
151
152 message_2 = customtkinter.CTkLabel(master=frame_22, justify=customtkinter.LEFT, text="Message à verifier :")
153 message_2.place(x=10, y=10)
154
155 msg_2 = customtkinter.CTkTextbox(master=frame_22, width=350, height=70)
156 msg_2.place(x=10, y=40)
157
158
159 hashage_2 = customtkinter.CTkLabel(master=frame_22, justify=customtkinter.LEFT, text="Hash (*) :")
160 hashage_2.place(x=10, y=120)
161

```

```

interface.py > hashing2
162
163 hash2_value = customtkinter.StringVar()
164 hash_2 = customtkinter.CTkEntry(master=frame_22, width=350, textvariable=hash2_value)
165 hash_2.place(x=10, y=150)
166
167
168 button_generer2 = customtkinter.CTkButton(master=frame_22, text="Générer", width=100, command=hashing2)
169 button_generer2.place(x=400, y=10)
170
171 #####
172
173 button_copier = customtkinter.CTkButton(master=frame, text="Copier", command=copier)
174 button_copier.place(x=300, y=500)
175
176 button_verify = customtkinter.CTkButton(master=frame, text="Vérifier", command=verify)
177 button_verify.place(x=450, y=500)
178
179 button_close = customtkinter.CTkButton(master=frame, text="Terminer", command=close_window)
180 button_close.place(x=600, y=500)
181
182 app.mainloop()

```