Université Mohammed V

Ecole Nationale Supérieure d'Informatique et d'Analyse des Systèmes

- ENSIAS -

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 :

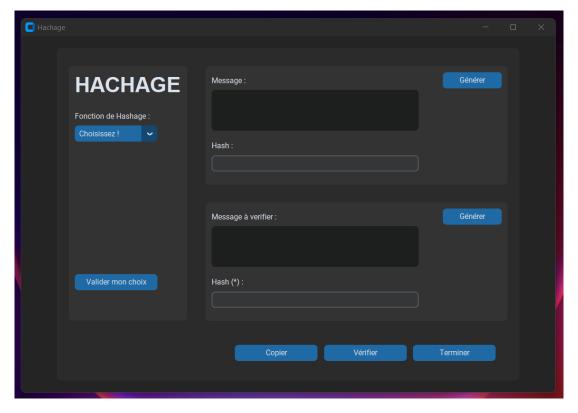
- MD5 (message-digest algorithm): c'est un algorithme de hachage développé en 1991 par Ronald L. Rivest et il a été spécifié en 1992 comme RFC 1321. Cet algorithme prend en entrée n'importe quelle taille de données numériques et produit une sortie de 128 bits. Cet algorithme est généralement utilisé, 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 hashs 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.

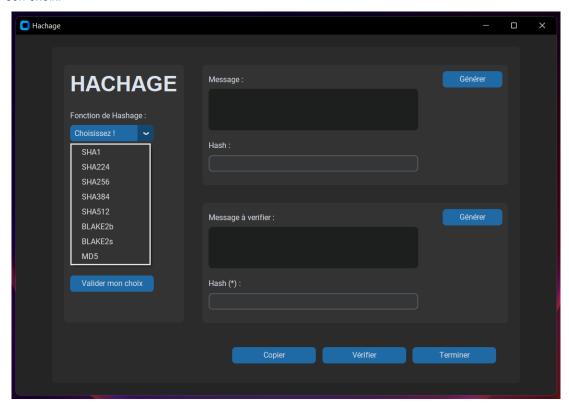
2 - Réalisation de l'interface :

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

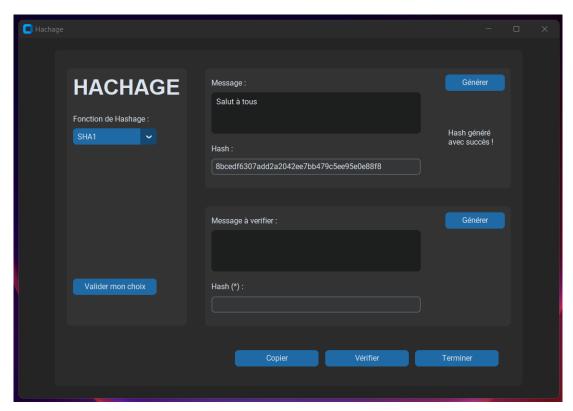
Cette interface nous permet de générer le hash d'une chaine de caractères données en entrée ainsi de le comparer avec un autre hash donné par l'utilisateur ou bien généré à partir de cette interface.



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.



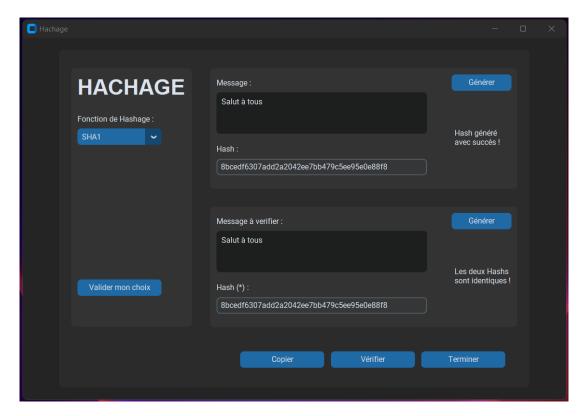
Dans la partie droite en haut, l'utilisateur peut saisir le message qui veut générer son hash en cliquant sur le bouton « générer ».

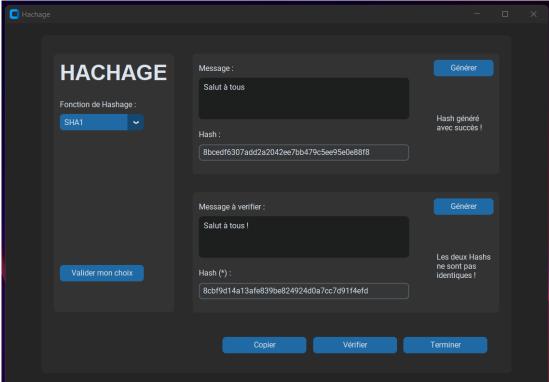


Ainsi qu'en bas, c'est exactement la même chose.

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 hashs s'ils sont identiques ou non.





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

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

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)

Annexe:

Notre interface codée en Python:

```
interface.py
    import customtkinter
    from tkinter import *
   import hashlib
   customtkinter.set_appearance_mode("dark") # Modes: "System" (standard), "Dark", "Light"
   customtkinter.set_default_color_theme("blue") # Themes: "blue" (standard), "green", "dark-blue"
   app = customtkinter.CTk()
   app.geometry("900x600")
   app.title("Hachage")
   frame = customtkinter.CTkFrame(master=app, width=900, height=600)
   frame.pack(pady=20, padx=60, expand=True)
   frame_1 = customtkinter.CTkFrame(master=frame, height=430)
   frame_1.place(x=20, y=30)
   frame_21 = customtkinter.CTkFrame(master=frame, width= 510)
   frame_21.place(x=250, y=30)
   frame_22 = customtkinter.CTkFrame(master=frame, width= 510)
   frame_22.place(x=250, y=260)
   def close_window():
       app.quit()
   def hash2(b):
       hash2 value.set(b)
    def hash1(b):
       hash1_value.set(b)
```

```
def get_value_choix():
    print(choix_function.get())

def input_msg_1():
    user_input = msg_1.get("1.0", "end").strip()
    print(user_input)

def input_msg_2():
    user_input = msg_2.get("1.0", "end").strip()
    print(user_input)

def copier():
    msg_2.delete("1.0", "end")
    msg_2.insert("1.0", msg_1.get("1.0", "end").strip())

def copier():
    msg_2.insert("1.0", msg_1.get("1.0", "end").strip())
```

```
def hashing1():
            f = choix_function.get()
           user_input = msg_1.get("1.0", "end").strip()
           if (f == 'SHA1') :
                result = hashlib.sha1(user_input.encode())
                result = hashlib.sha224(user_input.encode())
           if (f == 'SHA256'):
                result = hashlib.sha256(user_input.encode())
           if (f == 'SHA384'):
                result = hashlib.sha384(user_input.encode())
                result = hashlib.sha512(user_input.encode())
           if (f == 'BLAKE2b') :
                result = hashlib.blake2b(user input.encode())
           if (f == 'BLAKE2s') :
                result = hashlib.blake2s(user_input.encode())
           if (f == 'MD5'):
              result = hashlib.md5(user_input.encode())
          hash1(result.hexdigest())
          rep_1 = customtkinter.CTkLabel(master=frame_21, justify=customtkinter.LEFT, text="Hash généré \navec
          rep 1.place(x=405, y=100)
      def hashing2():
          f = choix_function.get()
          user_input = msg_2.get("1.0", "end").strip()
          if (f == 'SHA1') :
              result = hashlib.sha1(user input.encode())
          if (f == 'SHA224')
              result = hashlib.sha224(user_input.encode())
              result = hashlib.sha256(user_input.encode())
          if (f == 'SHA384') :
              result = hashlib.sha384(user input.encode())
          if (f == 'SHA512') :
              result = hashlib.sha512(user_input.encode())
              result = hashlib.blake2b(user_input.encode())
          if (f == 'BLAKE2s')
              result = hashlib.blake2s(user input.encode())
          if (f == 'MD5'):
              result = hashlib.md5(user_input.encode())
          hash2(result.hexdigest())
      def verify() :
          if len(hash1\_value.get())!=0 and len(hash2\_value.get())!=0 :
interface.py >  hashing2
              if(hash1_value.get() == hash2_value.get()) :
                   rep_2 = customtkinter.CTkLabel(master=frame_22, justify=customtkinter.LEFT, text="Les deux
                  rep_2 = customtkinter.CTkLabel(master=frame_22, justify=customtkinter.LEFT, text="Les deux
              rep_2.place(x=405, y=100)
      has has e = {\tt customtkinter.CTkLabel (master-frame\_1, justify={\tt customtkinter.LEFT, text="HACHAGE", font=the custom twinter.left"}) and the {\tt custom tkinter.LEFT, text="HACHAGE", font=the custom tkinter.left"} and {\tt custom tkinter.left"}.
      hashage.place(x=10, y=10)
      choix = customtkinter.CTkLabel(master=frame_1, justify=customtkinter.LEFT, text="Fonction de Hashage :")
      choix.place(x=10, y=70)
      choix_function = customtkinter.StringVar()
      hash_function = customtkinter.CTkOptionMenu(frame_1, values=["SHA1", "SHA224", "SHA256", "SHA384",
       "SHA512", "BLAKE2b", "BLAKE2s", "MD5"], variable=choix_function)
      hash_function.place(x=10, y=100)
      hash_function.set("Choisissez !")
      valider_choix = customtkinter.CTkButton(master=frame_1, text="Valider mon choix",
      command=get_value_choix)
      valider_choix.place(x=10, y=350)
```

```
message_1 = customtkinter.CTkLabel(master=frame_21, justify=customtkinter.LEFT, text="Message :")
     message_1.place(x=10, y=10)
🕏 interface.py > 🛇 hashing2
     msg1 = customtkinter.StringVar()
     msg_1 = customtkinter.CTkTextbox(master=frame_21, width=350, height=70)
     msg_1.place(x=10, y=40)
     hashage 1 = customtkinter.CTkLabel(master=frame 21, justify=customtkinter.LEFT, text="Hash :")
     hashage_1.place(x=10, y=120)
     hash1_value = customtkinter.StringVar()
     hash_1 = customtkinter.CTkEntry(master=frame_21, width=350, textvariable=hash1_value)
     hash_1.place(x=10, y=150)
     button_generer1 = customtkinter.CTkButton(master=frame_21, text="Générer", width=100, command=hashing1)
     button_generer1.place(x=400, y=10)
      message_2 = customtkinter.CTkLabel(master=frame_22, justify=customtkinter.LEFT, text="Message à verifier
     message_2.place(x=10, y=10)
     msg_2 = customtkinter.CTkTextbox(master=frame_22, width=350, height=70)
     msg_2.place(x=10, y=40)
     hashage_2 = customtkinter.CTkLabel(master=frame_22, justify=customtkinter.LEFT, text="Hash (*) :")
     hashage_2.place(x=10, y=120)

♦ interface.py > ♦ hashing2
     hash2_value = customtkinter.StringVar()
     hash_2 = customtkinter.CTkEntry(master=frame_22, width=350, textvariable=hash2_value)
     hash_2.place(x=10, y=150)
     button_generer2 = customtkinter.CTkButton(master=frame_22, text="Générer", width=100, command=hashing2)
     button_generer2.place(x=400, y=10)
     button_copier = customtkinter.CTkButton(master=frame, text="Copier", command=copier)
     button_copier.place(x=300, y=500)
     button_verify = customtkinter.CTkButton(master=frame, text="Vérifier", command=verify)
     button_verify.place(x=450, y=500)
     button_close = customtkinter.CTkButton(master=frame, text="Terminer", command=close_window)
     button_close.place(x=600, y=500)
```

app.mainloop()