FRANKFURT UNIVERSITY OF APPLIED SCIENCES
IM MODUL BETRIEBSSYSTEME UND RECHNERNETZE
BEI PROF. DR. CHRISTIAN BAUN
SOMMERSEMESTER 2021

Ausarbeitung Werkstück A

ALTERNATIVE 4: PASSWORT-MANAGER

VON MARC JAJONEK, CEDRIC KLUG-OFFERMANN
UND SOPHIE BINGNET

28. JUNI 2021

Inhaltsverzeichnis

1 Einleitung **1**

2 Problem **1**

3 Allgemeines **1**

4 Hauptprogramm (passman.py) **2**

5 Checks-Befehl **2**

6 Add-Befehl **3**

7 Delete-Befehl **4**

8 Change-Befehl **5**

9 Change_mp-Befehl **6**

10 Copy-Befehl **7**

11 Table-Befehl **8**

12 Datetime_check-Befehl **9**

13 Verentschlüsselung **9**

14 Genpassword-Funktion **11**

15 Zusatzfunktionen **12**
15.1 Password-Funktion 12
15.2 Help-Befehl 12
15.3 Datetime_check Benachrichtigung 12

16 Ergebnis **13**

17 Fazit **13**

18 Anhang **16**

Zusammenfassung Durch die zunehmende Digitalisierung ist es aus Sicherheitsgründen unabdingbar, immer häufiger verschiedene Passwörter zu verwenden. Je länger und abstrakter die Passwörter hierbei sind, desto sicherer sind diese. Das Ziel des vorliegenden Projektes ist es, verschiedene Passwörter zu speichern und zu organisieren. Um dies zu erreichen, arbeitet das Programm mit Benutzereingaben über Kommandozeilenargumente im Terminal. Der Benutzer hat hierbei die Möglichkeit über vorgegebene Befehle seine Passwörter zu verwalten.

1 Einleitung

„Der durchschnittliche Deutsche hat über 78 Online-Konten“ [1] - Zur Verwaltung, Generierung und Speicherung dieser Passwörter ist das folgende Projekt entstanden. Die Dokumentation geht auf die Vorgehensweise sowie die Implementierung in Python genauer ein. Das Projekt umfasst verschiedene Funktionen, die der Benutzer über das Terminal eingeben kann, wie zum Beispiel „add“, mit dem der Benutzer Passwörter im Passwort-Manager speichern kann. Die Anforderungen des Projektes sind zur besseren Übersicht in einem Ablaufdiagram dargestellt (siehe Anhang, Abbildung 1). Zunächst geht die Dokumentation auf das Problem und den dazugehörigen Lösungsweg genauer ein. Danach folgt eine detaillierte Beschreibung der Befehle sowie ihre Funktionsweisen anhand des Quellcodes und Beispieleingaben des Benutzers im Terminal. Abschließend folgt ein Fazit.

2 Problem

Im heutigen Digitalen Zeitalter benötigen Benutzer in ihrem täglichen Leben verschiedene Passwörter. Um eine größt mögliche Sicherheit zu gewährleisten, sollten sie so lang und komplex wie möglich sein. Außerdem sollte der Nutzer unterschiedliche Passwörter verwenden. Diese Situation erschwert es dem Nutzer, sich alle Passwörter zu merken und zu verwalten. Ein Passwort-Manager kann hierbei Abhilfe schaffen.

3 Allgemeines

Das folgende Projekt ist in einer UNIX-Umgebung entwickelt und setzt voraus, dass Python auf dem System installiert ist. Das Programm lässt sich über das Terminal bedienen. Hierfür muss der Nutzer bestimmte Befehle eingeben. Diese sind in der Dokumentation genauer beschrieben. Die Passwörter sind in einer Text-Datei gespeichert. Die hierfür benötigte Datei legt das Programm automatisch an. Sie befindet sich unter dem Pfad „\home\user” und ist für den Benutzer nicht sichtbar.

4 Hauptprogramm (passman.py)

Die Datei „passman.py", die der Benutzer im Terminal aufruft, ist das Hauptprogramm des ganzen Projektes. Es ruft je nach Eingabe die entsprechenden Funktionen auf. Die meisten Funktionen stehen in einem Try-Block geschrieben. Dieser soll zum einen den Error „KeyboardInterrups" abfangen, der entsteht, wenn der Nutzer das Programm z.B. durch Strg+C abbricht. Das Abfangen dieses Fehlers hat rein optischen Nutzen. Zum anderen fängt es den Error „IndexError" ab. Dieser Fehler tritt auf, wenn der Nutzer einen Befehl falsch im Terminal eingibt, und deshalb die erzeugte Liste sys.argv[] kürzer ist als erwartet. Die Liste sys.argv[] besteht aus der Eingabe des Nutzers im Terminal.

```
if sys.argv[1] == "help":
    help_Befehl()
else:
    try:
        check_Befehl()
        #An Hand der vom Nutzer eingegebenen Parameter wird entschieden,
             welche Methode aufegrufen wird
        #Add_Befehl wird aufgerufen wenn User add uebergibt
        if sys.argv[1] == "add":
            add_Befehl()
        #delete_Befehl wird aufgerufen wenn User delete uebergibt
        elif sys.argv[1] == "delete":
            delete_Befehl()
        #change_Befehl wird aufgerufen wenn User change uebergibt
        elif sys.argv[1] == "change":
            change_Befehl()
        #table_Befehl wird aufgerufen wenn User table uebergibt
        elif sys.argv[1] == "table":
            table_Befehl()
        #copy_Befehl wird aufgerufen wenn User copy uebergibt
        elif sys.argv[1] == "copy":
            copy_Befehl()
        #changemp wird aufgerufen wenn User changemp uebergibt
        elif sys.argv[1] == "changemp":
            change_mp()
    #KeyboardInterrupt bei Strg+C
    except KeyboardInterrupt:
        print("Transaction canceled!")
        sys.exit()
    #IndexError bei zu kurzer Eingabe eines Befehls
    except IndexError:
            print("Please enter your command in correct Syntax!\n"
                "Call Function help, if you need an example")
```

Listing 1: Aufruf der Befehle

5 Checks-Befehl

Die Funktion „check_Befehl()" führt das Programm bei fast jedem Programmdurchlauf aus. Sie überprüft, ob die Datenumgebung, also der Ablageordner „passman" und die Ablagedatei „pw_list.txt" bereits existieren. Je nach Gegebenheit, erstellt die Funktion den Ordner oder die Datei. Den Ordner legt die Funktion unter dem Benutzerverzeichnis (\home\user) an. Die Funktion erstellt den Ordner so, dass er nicht sichtbar für den Nutzer ist. Dies ist realisiert durch das Setzten eines Punktes vor dem Ordnernamen.

```
def check_Befehl():
    # Nutzernamen in variable Speichern
    user = getpass.getuser()
    # Ordner in den die PWList gespeichert werden soll in variable
        speichern
    ordner = Path("./.passman")
    file = Path("./pw_list.txt")
```

Listing 2: Deklaration der Variablen

Außerdem fordert die Funktion den Nutzer auf, sein Master-Password zu setzten. Das Master-Passwort schreibt die Funktion anschließend verschlüsselt in die Datei „pw_list.txt“. Zum Schluss fügt die Funktion den aktuellen Zeitstempel in die Datei „pw_list.txt” hinzu. Durch diesen realisiert das Programm die Zeitüberschreitung. Der einzige Befehl, der diese Funktion nicht aufruft, ist die Funktion „help_befehl()“. Diese muss unabhängig von der Datenumgebung dem Nutzer Informationen über die Benutzung des Passwortmanagers liefern. Deshalb überprüft das Programm bei diesem Befehl nicht, ob schon eine Datei bzw. ein Master-Passwort existiert.

6 Add-Befehl

Die Funktion „add()” ruft das Hauptprogramm auf, wenn der Nutzer ein neues Passwort anlegen möchte. Dies kann er tun, in dem er einen Befehl wie diesen, über das Terminal aufruft.

```
passman add -title TitleX -username UsernameX -generatepw
```

Listing 3: Beispiel add-Befehl im Terminal

Hierbei überprüft das Hauptprogramm, ob der Nutzer den Parameter „add“ übergeben hat. Anschließend ruft das Programm die Funktion „add()” auf. Diese Funktion bietet dem Nutzer zwei verschiedene Wege ein Passwort zu erstellen. Zum einen kann der Benutzer ein Passwort vom Programm generieren lassen. Dies ist möglich, in dem er den oben genannten Befehl aufruft. Es folgen Abfragen, die der Benutzer im Terminal beantworten muss. Hierbei kann er entscheiden, wie lang sein Passwort sein soll. Außerdem kann er auswählen, welche Zeichen das Passwort enthalten soll. Zur Auswahl stehen Zahlen, Kleinbuchstaben, Großbuchstaben oder Symbole. Da das Programm die Sicherheit des Passwortes gewährleisten muss, sollte das Passwort eine Länge von mind. 8 Zeichen haben und der Nutzer muss mind. 3 Zeichenkategorien bestätigen. Die Funktion speichert im Anschluss den Titel mit dem Benutzernamen in die Datei. Außerdem speichert das Programm das generierte Passwort verschlüsselt in die Datei. Die Verschlüsselung erfolgt mit Hilfe der Funktion „verschluesselung()”. Die andere Option, die die Funktion bietet ist, dass der Nutzer sein Passwort selber vergeben kann. Hierfür muss er im oben genannten Beispielbefehl -generatepw mit -password austauschen. Über das Terminal erscheint eine Meldung, dass den Nutzer auffordert sein Passwort zu vergeben. Durch die Funktion „getpass” ist

die Eingabe im Terminal nicht zu sehen. Das Passwort muss entsprechend sicher gestaltet sein. Deshalb muss der Nutzer ein Passwort wählen, welches Zahlen, Kleinbuchstaben, Großbuchstaben und Symbole enthält. Außerdem muss es eine Länge von mind. 8 Zeichen aufweisen. Die Sicherheit des Passwortes überprüft die Funktion „password()". Entspricht das Passwort nicht den Anforderungen, bekommt der Nutzer eine entsprechende Fehlermeldung im Terminal ausgegeben. Ist das Passwort sicher, speichert das Programm, den Titel, den Benutzernamen und das Passwort verschlüsselt ab.

7 Delete-Befehl

Die Funktion „delete()" ruft das Hauptprogramm auf, wenn der Nutzer ein Passwort löschen möchte. Dies kann er tun, in dem er einen Befehl wie diesen über das Terminal aufruft.

```
passman.py delete -title TitelX
```

Listing 4: Beispiel delete-Befehl im Terminal

Hierbei überprüft das Hauptprogramm, ob der Nutzer den Parameter „delete" übergeben hat. Anschließend ruft das Programm die Funktion „delete()" auf. Die Funktion dient dazu, alle Einträge, die zu einem Titel gehören zu löschen. Die Funktion überprüft als erstes, ob der angegebene Titel überhaupt existiert. Existiert dieser nicht, bekommt der Nutzer eine entsprechende Fehlermeldung. Existiert der Titel, fragt das Programm den Nutzer, ob er die Einträge sicher löschen möchte. Durch die erneute Abfrage ist ein irrtümliches Löschen ausgeschlossen. Die Funktion sucht nach einer Übereinstimmung von einem Eintrag in der Datei „pw_list.txt" zum eingegebenen Titel aus dem Terminal. Ist die Übereinstimmung gefunden, kann das Programm von dort aus, mittels der Funktion „pop()", Einträge aus einer Liste löschen.

```
#Wenn ja, dann wird der komplette Eintrag zum eingegebenen Title
    geloescht
if loeschen == "y":
    text_arr.pop(i+5)
    text_arr.pop(i+4)
    text_arr.pop(i+3)
    text_arr.pop(i+2)
    text_arr.pop(i+1)
    text_arr.pop(i)
    text_arr.pop(i-1)
```

Listing 5: Einträge löschen

Dadurch löscht das Programm alle Einträge, die zu dem eingegebenen Titel gehören. Um die Änderung in der Datei „pw_list.txt" abzuspeichern, beschreibt das Programm die Datei mit der bearbeiteten Liste neu. Somit sind die Einträge gelöscht.

```
a_file = open("pw_list.txt", "w")
j = 0
#Mittels Schleife wird die Datei komplett neugeschrieben, ohne die
    geloeschten Elemente
while j < anzahl:
        #Das letzte \n soll damit wegfallen, da sonst die Datei falsch
            weitergeschrieben wird.
        if j == len(text_arr)-1:
              a_file.write(text_arr[j])
              break
        else:
              a_file.write(text_arr[j]+" ")
        j += 1
a_file.close()
break
```

Listing 6: Datei neu schreiben

8 Change-Befehl

Die Funktion „change_befehl()" ruft das Hauptprogramm auf, wenn der Nutzer ein Passwort ändern möchte. Dies kann er tun, in dem er einen Befehl wie diesen, über das Terminal aufruft.

```
passman.py change -title TitelX -generatepw
```

Listing 7: Beispiel change-Befehl im Terminal

Hierbei überprüft das Hauptprogramm, ob der Nutzer den Parameter „change" übergeben hat. Anschließend ruft das Programm die Funktion „change_befehl()" auf. Die Funktion überprüft als erstes, ob schon ein Eintrag in der Datei „pw_list.txt" existiert. Wenn es keinen Eintrag gibt, kann das Programm auch kein Passwort ändern. Somit bekommt der Nutzer eine Fehlermeldung ausgegeben. Außerdem überprüft die Funktion, ob es eine Übereinstimmung in der Datei zum eingegebenen Titel gibt. Diese Voraussetzung muss auch erfüllt sein. Ist sie nicht erfüllt, bekommt der Nutzer eine Fehlermeldung ausgegeben. Im Anschluss überprüft die Funktion, auf welche Art der Nutzer sein Passwort ändern möchte. Hierbei gibt es wieder zwei verschiedene Möglichkeiten. Zum einen kann der Nutzer das Passwort generieren lassen und zum anderen kann er das Passwort selbst vergeben. Um das Passwort mittels der Funktion „genpassword()" generieren zu lassen, muss der Nutzer den Befehl wie oben angegeben im Terminal eingeben. Die Funktionsweise der Funktion „genpassword()" ist in der Dokumentation genauer beschrieben. Um das Passwort im Terminal selbst vergeben zu können, muss der Nutzer im Befehl -generatepw durch -password ersetzen. Im Anschluss fordert das Programm den Nutzer auf, sein Passwort zu vergeben. Das Programm überprüft das Passwort mittels der Funktion „password()" auf Sicherheit. Die Funktion „passsword()" ist in der Dokumentation genauer beschrieben. Das neue Passwort schreibt das Programm in beiden Fällen gleich in die Datei. Die Funktion geht von der Stelle der Übereinstimmung der Titel zum dazugehörigen Passwort und überschreibt dieses mit dem neuen Passwort. Das Passwort verschlüsselt die Funktion „verschluesslung()" vorher. Anschließend

muss das Programm die Datei neu beschreiben, damit die Datei „pw_list.txt" die Änderungen enthält. Dies geschieht, indem die Funktion die Datei mit allen Einträgen samt des geänderten Passwortes erneut beschreibt.

```
b_file = open("pw_list.txt", "w")
j = 0
#Mittels Schleife soll das text_arr durchgegangen werden und die Stelle finden, an der der eingebene Title dem Title aus der Datei gleicht.
while j < anzahl:
     if sys.argv[3] == text_arr[j]:
          #Aendern des Passworts in das neue Passwort
          text_arr[j+4] = verschluesselung(gen_pw)
          k=0
          #Mittels Schleife wird die Datei komplett neu geschrieben, somit wird die Aenderung mitgenommen.
          while k < anzahl:
               #Das letzte Leerzeichen soll damit wegfallen, da sonst die Datei falsch weitergeschrieben wird.
               if k == anzahl-1:
                    b_file.write(text_arr[k])
                    break
               else:
                    b_file.write(text_arr[k]+" ")
                    k += 1
     j +=1
```

Listing 8: neu Beschreibung der Datei

9 Change_mp-Befehl

Die „change_mp" Methode befasst sich mit der Änderung des Master-Passworts. Das Programm fordert den Nutzer dazu auf, ein neues Passwort einzugeben. Dabei prüft die Methode "password()" die Sicherheit der Eingabe. Ist die Sicherheit sichergestellt, verschlüsselt die Methode "verschluesselung()" das neue Passwort und speichert es in der "pw_list.txt"-Datei ab. Sollte bei der Eingabe ein Fehler bei dem Befehl vorliegen, so beendet das Programm den Vorgang und gibt eine Fehlermeldung aus.

```
password_var = getpass.getpass("Please enter your new master-password: \n")
#Prueft ob das eingegebene Password sicher genug ist und unseren standards entspricht
if password(password_var) == True:
    i = 0
    while i < anzahl:
        if text_arr[i] == "MasterPassword":
            text_arr[i+1] = verschluesselung(password_var)
        #Neuschreiben der Liste
        k = 0
        a_file = open("pw_list.txt", "w")
        while k < anzahl:
            if k == anzahl-1:
                a_file.write(text_arr[k])
                break
            else:
                a_file.write(text_arr[k] + " ")
            k +=1
        i += 1
```

Listing 9: Masterpasswort ändern

10 Copy-Befehl

Damit der Benutzer seine gespeicherten Passwörter bei Bedarf verwenden kann, kopiert der Copy-Befehl diese für 30 Sekunden in die Zwischenablage. Dadurch sind die Passwörter nie im Klartext im Terminal zu sehen. Der Benutzer kann diese somit bei einer Anmeldung unkompliziert einfügen .

```
passman copy -title TitleX
```

Listing 10: Beispiel copy-Befehl im Terminal

Um die Passwörter zusätzlich zu sichern, verlangt das Programm nach Funktionsaufruf zunächst nach einer Master-Passwort Eingabe. Dies soll verhindern, dass Unbefugte innerhalb der 5min Master-Passwort Abfrage, Zugriff auf die Passwörter haben. Danach liest das Programm Titel, Benutzername und Passwort in eine Liste ein. Mithilfe einer If-Abfrage überprüft das Programm, ob der eingebende Titel in der Liste gespeichert ist. Wenn der Titel in der Liste gespeichert ist, speichert die variable „title_pos“, die Position an der das Passwort gespeichert ist. Dies ist notwendig, um Zugriff auf den Username und das Passwort zu haben, da diese hintereinander in der Liste gespeichert sind. So kann das Programm bei einer Ausgabe im Terminal, den Benutzernamen mit ausgeben.

```
1  if sys.argv[3] in data_arr:
2      title_pos = data_arr.index(sys.argv[3])
3      # Username befindet sich im
4      y an der Position nach Title
5      print("Username: " + data_arr[title_pos + 1] + " | password copied to
           clipboard")
6      start = datetime.datetime.now()
7
8      # nicht laenger als 30 sec in Clipboard speichern
9      while(timedelta.total_seconds(datetime.datetime.now() - start)) <= 30:
10         # Passwort befindet sich im Array zwei Positionen nach Title
11         pyperclip.copy(entschluesselung(data_arr[title_pos + 2]))
12     pyperclip.copy("")
13     print("30 sec are over, deleted password from clipboard!")
14 else:
15     print("Title doesn't exist")
```

Listing 11: Passwort in Zwischenablage speichern

Nach der Ausgabe speichert die Variable „start“ die aktuelle Zeit (Zeile 6). Eine while-Schleife überprüft, mithilfe der Funktion „timedelta.totalseconds“, ob die aktuelle Zeit minus die Startzeit kleiner oder gleich 30 Sekunden ist. Falls ja, speichert das Programm das Passwort, mittels der Pyperclip-Funktion in der Zwischenablage. Der Benutzer muss vorher die Pyperclip-Funktion installieren[1]. Während der 30 Sekunden steckt das Programm in der while-Schleife. Der Benutzer kann also keine weiteren Kommandozeilen-Befehle aufrufen. In dieser Zeit sollte man in der Regel jedoch damit beschäftigt sein, dass Passwort an der passenden Stelle einzufügen. Nachdem das Programm aus der while-Schleife herauskommt und die 30 Sekunden vorbei sind, kopiert es einen leeren String

[1] https://pypi.org/project/pyperclip/

in die Zwischenablage und gibt aus, dass die Zeit vorbei ist und das Passwort gelöscht ist (Zeile 12-13).

11 Table-Befehl

Durch den Table-Befehl ist es für den Benutzer möglich alle Titel und die dazugehörigen Benutzernamen, die im Passwort-Manager gespeichert sind, im Terminal ausgegeben zu bekommen. Die Funktion lässt sich durch den Kommmandozeilen-Befehl „table“ aufrufen.

```
passman table
```

Listing 12: Beispiel table-Befehl im Terminal

Nach einem Befehlaufruf erstellt das Programm die Liste „tabledata_arr“ . In dieser Liste befinden sich alle Informationen, die das Programm in der Tabelle ausgibt. Um zusammengehörige Einträge (Titel und Nutzername) in einer Zeile im Terminal ausgeben zu können, muss das Programm diese in einer separaten Liste speichern(splitdata_arr). Mittels einer for-Schleife geht das Programm die gesamte Liste „tabledata_arr“ vom Startwert z und dem Endwert z+2 durch. Die Variable z steht für alle Einträge. Die Funktion „range“ gibt eine Sequenz der Liste vom Start 0 bis zur Länge der Liste an, sowie eine Erhöhung um den Wert 2 nach jedem Durchlauf.

```
splitdata_arr = [
     tabledata_arr[z:z +2]
     for z in range(0,len(tabledata_arr), 2)
]
```

Listing 13: Array auf Größe 2 trennen

Die Funktion „fiexed_lenght“ sorgt dafür, dass alle Tabellenspalten in der Ausgabe dieselbe Größe haben. Die Funktion benötigt hierfür den zu formatierenden Text, sowie die Spaltenbreite. Sie überprüft, ob der gegebene Text größer als die angegebene Länge ist oder kleiner. Wenn dieser kleiner ist, hängt sie Leerzeichen bis zur Erreichung der Länge an den Text an. Bei größeren Texten schneidet die Funktion den Text an der vorgegebenen Länge ab.

```
#Formatierung der Tabellenspalten
def fixed_lenght(text,lenght):
     if len(text) > lenght:
         text = text[:lenght]
     elif len(text) < lenght:
         text = (text + " " * lenght) [:lenght]
     return text
```

Listing 14: Formatierung der Tabellenspalten

Um die Tabelle übersichtlich im Terminal auszugeben, trennt das Programm die Einträge durch „—“ und die Spalten durch „-„. In der ersten Tabellenspalte stehen die Zeilenbezeichnungen, die in der Liste „header“ gespeichert sind. Das Programm geht alle Indizes (column) durch und gibt diese aus. Das Programm übergibt die Einträge der Funktion „fixed_lenght“ als Text. Um alle Einträge

des „splitdata_array“ ausgeben zu können, muss eine for-Schleife zunächst durch alle Zeilen (rows) gehen und in diesen dann durch eine weitere for-Schleife alle Indizes ausgeben.

12 Datetime_check-Befehl

Diese Methode befasst sich mit der Lokalisierung dem/der aktuellen Datum/Uhrzeit und dem/der Datum/Uhrzeit, die sich in der “pw_list.txt” befindet, die bei der Erzeugung der “pw_list.txt” automatisch erfasst und gespeichert ist. Beim Eingeben des Master-Passworts prüft sie das Datum und die aktuelle Uhrzeit in der „pw_list.txt“. Befindet sich diese Zeitspanne außerhalb eines 5-minütigen Zeitfenster, so muss der Benutzer das Master-Passwort erneut eingeben.

```
1  # Abfrage ob Datum in der Datei identisch ist und ob die Zeitliche
       differenz zwischen "Aktuell" und "Datei" kleiner ist als 5 Minuten
       (300 Sekunden)
2  if str(datum_datei) == str(datum_now) and (zeit_diff.seconds <= 300):
3      # Wenn das Datum uebereinstimmt und man sich im Zeitlichen Rahmen von
           30 Minuten befindet, wird noch abgefragt, ob die bereits
           vergangene Zeit
4      # groesser als 3 Minten ist (180 Sekunden)
5      if zeit_diff.seconds >= 180:
6          # Wenn man nur noch 3 Minten oder weniger zeit hat, wird die
               Volle Minute zusammen mit einem Text ausgegeben, der besagt,
               wie viel zeit man
7          # noch hat, bis man das Passwort wieder erneut eingeben muss
8          zeit_diff_minute = round((300 - zeit_diff.seconds) / 60)
9          print("You have " + str(zeit_diff_minute) + " minutes left")
10     else:
11         print("Master password: OK")
```

Listing 15: Abfrage zur verbleibenden Zeit

Um den Nutzer eine Übersicht der verbleibenden Zeit zu ermöglichen, folgt eine Abfrage, ob die verbleibende Zeit 3 Minuten oder weniger beträgt (siehe Kapitel 15.3).

13 Verentschlüsselung

In der Verentschlüsselung befasst sich das Programm mit dem Ver- und Entschlüsseln der Passwörter. Dies dient der Sicherheit und orientiert sich an der Caesar-Verschlüsselung

Verschlüsselung

Die Methode „verschluesselung()“ dient dazu, Benutzereingaben zu verschlüsseln. Dazu verwendet die Methode die sogenannte “Caesar-Verschlüsselung”. Um diese Verschlüsselung anzuwenden, muss sich das Alphabet um einen bestimmten Wert verschieben. Der Einfachheit halber beträgt der Wert in diesem Programm den Wert 1. Somit macht die Verschlüsselung aus dem Buchstaben “A” ein “B”, aus dem “B” ein “C” und dies geht dann bis zum Buchstaben “Z” aus dem dann ein “A” entsteht. Hierbei liest die Methode den jeweiligen Charakter und prüft,

ob es sich um eine Zahl, einen Kleinbuchstaben, einen Großbuchstaben oder ein Sonderzeichen handelt. Je nachdem verwendet das Programm eine andere Mathematische Formel, um sicher zu stellen, dass es zu keinem Error kommt und um es entschlüsseln zu können. Nach erfolgreichem verschlüsseln der einzelnen Charaktere, fügt die Methode diese zusammen und speichert das Passwort in der "pw_list.txt".

```
1  def verschluesselung(usereingabe):
2      liste = []
3      # Kleinbuchstarben werden rausgesucht
4      klein = string.ascii_lowercase
5      # Grossbuchstareben werden rausgesucht
6      gross = string.ascii_uppercase
7      # Zahlen werden rausgesuch
8      zahl = string.digits
9      # Sonderzeichen werden rausgesuch
10     zeichen = string.punctuation
11     # Iteration durch den Text des users zur Verschluesselung - Character
           fuer Character
12     for i in range(len(usereingabe)):
13         # Pruefung ob aktueller Character ein Kleinbuchstabe ist
14         if usereingabe[i] in klein:
15             index = klein.find(usereingabe[i])
16             # Caesar-Verschluesselung um einen bestimmten Wert
17             umwandlung = klein[(index + 1)%26]
18             liste.append(umwandlung)
19         # Selbes fuer Grossbuchstabe
20         elif usereingabe[i] in gross:
21             index = gross.find(usereingabe[i])
22             umwandlung = gross[(index + 1)%26]
23             liste.append(umwandlung)
24         # Selbes fuer Zahlen
25         elif usereingabe[i] in zahl:
26             index = zahl.find(usereingabe[i])
27             umwandlung = zahl[(index + 1)%10]
28             liste.append(umwandlung)
29         # Selbes fuer Sonderzeichen
30         elif usereingabe[i] in zeichen:
31             index = zeichen.find(usereingabe[i])
32             umwandlung = zeichen[(index + 1)%26]
33             liste.append(umwandlung)
34         else:
35             liste.append(usereingabe[i])
36     # Zusammensetzung der Verschluesselung und Rueckgabe
37     usereingabe = ''.join(liste[i] for i in range(len(liste)))
38     return usereingabe
```

Listing 16: Verschlüsselung

In Zeile 38 setzt die Methode die einzelnen Charaktere zusammen und somit entsteht das verschlüsselte Passwort.

Entschlüsselung

In der Entschlüsselung verringert das Programm den Wert der einzelnen Charaktere um 1. Demnach entschlüsselt die Funktion bspw. aus dem Buchstaben "B" ein "A". Auch hier nimmt die Methode jeden einzelnen Charakter, verringert den Wert und setzt es zusammen. Diese Anwendung findet seinen Zweck bspw. in der "copy" Methode. Denn dort muss das entschlüsselte Passwort, zur Weiterverarbeitung, in die Zwischenablage gespeichert sein.

```
if usereingabe[i] in klein:
    index = klein.find(usereingabe[i])
    # Caesar-Entschluesselung um einen bestimmten Wert
    umwandlung = klein[(index - 1)%26]
    liste.append(umwandlung)
```

Listing 17: Entschlüsselung

14 Genpassword-Funktion

Die Funktion „genpassword()" ist eine Hilfsfunktion. Sie ist für die Generierung von Passwörtern zuständig. Die Funktion führt mehrere Abfragen durch, durch die der Nutzer bestimmen kann, aus welchen Zeichen sein Passwort bestehen soll. Aus den akzeptierten Zeichenkategorien erstellt die Funktion dann ein sicheres Passwort. Die Funktion fragt den Nutzer als erstes, welche Länge das Passwort haben soll. Hierbei muss der Nutzer eine Länge von mindestens 8 eingeben. Danach folgen Abfragen, die der Nutzer mit „y" beantworten muss, um diese zu bestätigen. Die Abfragen beinhalten, ob das Passwort die Zeichenkategorie beinhalten soll. Der Nutzer hat Nummer, Kleinbuchstaben, Großbuchstaben und Symbole zur Auswahl. Er muss mindestens drei der vier Kategorien akzeptiert haben. Hat er dies nicht, bekommt er eine Fehlermeldung ausgeben. Dies stellt sicher, dass das Passwort sicher ist. Die akzeptierten Zeichenkategorien, speichert die Funktion in einen String.

```
else:
      if small_bool == "y":
            lower = string.ascii_lowercase
            all += lower
      else:
            pass
      if capital_bool == "y":
            upper = string.ascii_uppercase
            all = all + upper
      else:
            pass
      if symbol_bool == "y":
            symbols = string.punctuation
            all = all + symbols
      else:
            pass
      if num_bool == "y":
            num = string.digits
            all = all + num
      else:
            pass
```

Listing 18: Abfragen zu den Zeichen des Passwortes

Durch die Funktion „random.sample()" erzeugt die Funktion eine Liste, die eine zufällige Auswahl an Zeichen, aller akzeptierten Zeichenkategorien, jeweils in einen Index schreibt.

```
temp = random.sample(all,pw_lenght)
#Variable fertigstellen
gen_pw = "".join(temp)
```

Listing 19: Zufällige Kombination von Zeichen

Aus dieser Liste erzeugt die Funktion mittels „join()" einen String. Der String ist das endgültig generierte Passwort und dient als Rückgabewert der Funktion.

15 Zusatzfunktionen

Die Zusatzfunktionen sollen Nutzern den Gebrauch des Passwort-Managers erleichtern und seine Funktionsweise verbessern.

15.1 Password-Funktion

Die Funktion „password()" ist eine Hilfsfunktion. Sie ist für die Überprüfung von Passwörtern zuständig. Die Funktion überprüft, ob selbst vergebene Passwörter sicher sind. Ist ein Passwort sicher, gibt es den Wert TRUE zurück. Die erste Abfrage überprüft, ob das Passwort mindestens 8 Zeichen lang ist. Außerdem überprüft die Funktion, ob das Passwort Zahlen, Kleinbuchstaben, Großbuchstaben und Symbole enthält. Da der Nutzer das Passwort selber vergibt, muss das Passwort alle Zeichenkategorien beinhalten. Die Überprüfung des Passwortes auf bestimmte Zeichen ist durch die Funktion „re.search()" realisiert.

```
#Ueberpruefen ob das Passwort Zahlen beinhaltet
elif re.search('[0-9]', password) is None:
    print("Make sure your password has a number in it.")
```

Listing 20: Überprüfung des Passwortes auf Zahlen

Durch „re.search()" überprüft die Funktion einen String auf bestimmte Zeichen. Enthält das Passwort eine Zeichenkategorie nicht, bekommt der Nutzer eine Fehlermeldung, in der das Programm auf die nicht erfüllte Zeichenkategorie hinweist.

15.2 Help-Befehl

Der Help-Befehl ist eingebaut, um Benutzern bei Eingabeproblemen zu helfen. Dieser gibt für jeden Befehl ein Beispiel an, wie Kommandozeilen-Befehle korrekt einzugegeben sind. Bei jedem Syntaxfehler verweist das Programm auf diesen Help-Befehl. Für eine übersichtlichere Ausgabe ist auch hier die „fixed_lenght" Methode implementiert (siehe Listing 14).

15.3 Datetime_check Benachrichtigung

Eine Zusatzfunktion ist die Benachrichtigung des Nutzers, über die verbleibende Zeit. Hat der Benutzer nur noch weniger als 3 Minuten, gibt die Methode beim Ausführen jeder anderen Methode, die verbleibende Zeit an. Ist die Zeit

ausgelaufen und der Nutzer konnte das Master-Passwort erfolgreich eingeben, so schreibt die Methode die aktuelle Uhrzeit und das aktuelle Datum in die „pw_list.txt". Somit ist sichergestellt, dass es zu keinen Komplikationen kommen kann, wenn sich der Nutzer zur selben Uhrzeit, aber an einen anderen Tag einloggt.

```
while k < len(text_arr):
    if text_arr[k] == "Zeit":
        datei.write("Zeit ")
        datei.write(str(text_arr[k+1]+" "))
        datei.write(str(text_arr[k+2]+ " "))
         k +=2
    #Alle anderen in der Datei verhandenen informationen werden auch neu
        geschrieben, da sie sonst geloescht werden wuerden
    else:
        datei.write(text_arr[k]+" ")
    k += 1
```

Listing 21: Speicherung der aktuellen Zeit

16 Ergebnis

Die Entwicklung mit allen Anforderungen ist erfolgreich verlaufen. Probleme gab es zu Beginn bei der Wahl der Programmiersprache, durch fehlende Kenntnisse der vorausgesetzten Programmiersprachen. Nach einer Abwägung der Programmiersprachen Python und Bash, stellte sich Python als passend heraus. Außerdem trat bei der Überprüfung der Master-Passwort Eingabe das Problem auf, dass das Programm einen eigentlichen Zeitstempel als String aus der Datei auslas. Bei der Weiterverarbeitung des Zeitstempels führte dies zu einem Konflikt der Datentypen. Das Problem lag daran, dass die Verrechnung eines Zeitstempels nicht mit einem String möglich war. Die Umwandlung des Strings in den Datentyp Datetime löste das Problem.
Wegen der Funktion „Pyperclip", die z.B. im „copy-Befehl()" Verwendung findet, geht das Programm davon aus, dass das Paket Pyperclip aus dem Paketverwaltungsprogramm „pip3" über das Terminal auf dem System installiert ist. Das Programm sieht eine Linux Distribution vor. Bei der Verwendung eines anderen Betriebssystems, wie z.B MacOs sind kleinere Änderungen am Skript Notwendig, da die Verzeichnissstrukturen der Betriebssyteme unterschiedlich sind.

17 Fazit

Der add Befehl dient zum Anlegen und Generieren von Passwörtern. Bereits angelegte Passwörter kann der Nutzer mit „change" ändern. Durch „delete" kann der Nutzer Einträge löschen. Das Kopieren eines Passwortes ist mithilfe von „copy" möglich. Somit ist das Programm ein vollständiger Passwort-Manager. Die Zusatzfunktionen help, der Sicherheits-Check von Passwörtern und die Benachrichtigung der Zeit vor der Wiedereingabe des Master-Passwortes dienen

als Erweiterungen des Programmes und runden dieses ab. Trotz anfänglicher Schwierigkeiten, entspricht das Projekt allen Anforderungen. Zukünftige Erweiterungen könnten beispielsweise eine GUI (graphical user interface) sein. Außerdem wäre eine verschlüsselte Ablagedatei ein zusätzlicher Sicherheitsfaktor. Die Passwörter könnten zudem in einer Datenbank gesichert sein. Die Organisation von Passwörtern ist mittels des Passwort-Managers für Nutzer leicht zu realisieren.

Literatur

[1] Bott, Georgina (2017, Mai 04). *Haben wir ein Passwortproblem?*
https://www.marconomy.de/haben-wir-ein-passwortproblem-a-605153/

[2] Python Software Foundation (2021, Juni 24). *Basic date and time types*
https://docs.python.org/3/library/datetime.html

[3] Johannes Ernesti, Peter Kaiser. *Python 3: Das umfassende Handbuch.*
Rheinwerk Verlag GmbH. 2020
https://openbook.rheinwerk-verlag.de/python/30_001.html#u30

[4] Doug Hellmann (2020, Juli 11). *getpass – Prompt the user for a password without echoing*
https://pymotw.com/2/getpass/

[5] Ayushi Rawat (2020, Nov 30). *Create a Random Password Generator using Python*
https://medium.com/analytics-vidhya/create-a-random-password-generator-using-python-2fea485e9da9

18 Anhang

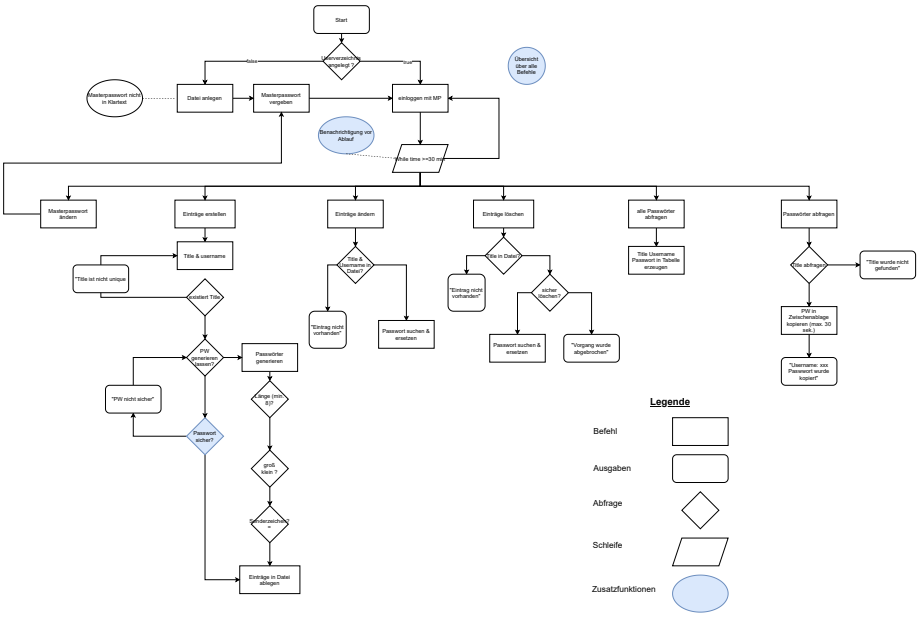

Abbildung 1: Projektplan