(Titelblatt)

# Inhaltsverzeichnis

[Inhaltsverzeichnis 2](#_Toc198553030)

[Versionsverlauf 5](#_Toc198553031)

[Kontaktdaten 6](#_Toc198553032)

[Lastenheft 7](#_Toc198553033)

[Pflichtenheft 8](#_Toc198553034)

[Struktogramm 9](#_Toc198553035)

[Funktionsdefinitionen 10](#_Toc198553037)

[Funktionsbeschreibung 10](#_Toc198553038)

[Code 11](#_Toc198553039)

[Testing 16](#_Toc198553040)

# Versionsverlauf

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Ersteller** | **Anpassung** | **Datum** |
| 0.1 | Janis | - | 19.05.2025 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# 

# Kontaktdaten

Name: Janis Bruhin

Tel. Nummer: 077 501 37 41

Email: janis.bruhin@stud.bzbs.ch

# Lastenheft

* Sinnvolles Projekt mithilfe vom Bash programmieren.

Ziel dieses Projekts ist es, ein einfaches Passwort-Manager-Skript zu entwickeln, das:

* in der Kommandozeile ausführbar ist
* Passwörter generieren und sicher abspeichern kann
* einen Master-Passwortschutz enthält
* bereits gespeicherte Passwörter anzeigen kann
* einfach zu bedienen ist
* keine externe Software (ausser Standard-Tools wie OpenSSL) benötigt

# Pflichtenheft

Das Script muss folgende Anforderungen erfüllen:

* Beim ersten Start wird der Benutzer zur Einrichtung eines Master-Passworts aufgefordert
* Die Passwörter werden verschlüsselt mit AES-256 im lokalen Dateisystem gespeichert
* Das Master-Passwort wird als SHA-256 Hash gespeichert (nicht im Klartext)
* Es gibt ein Menü mit den Optionen:
  1. Master-Passwort setzen
  2. Passwort hinzufügen
  3. Passwörter anzeigen
  4. Programm beenden
* Fehlerbehandlung bei falschem Master-Passwort
* Optional generierte oder manuell eingegebene Passwörter

# Struktogramm

# Ein Bild, das Text, Screenshot, Diagramm, Reihe enthält. KI-generierte Inhalte können fehlerhaft sein.

# Funktionsdefinitionen

|  |  |
| --- | --- |
| generate\_password(length) | Generiert ein zufälliges Passwort mit angegebener Länge |
| set\_master\_password() | Setzt ein neues Master-Passwort, speichert den Hash |
| check\_master\_password() | Prüft, ob das eingegebene Passwort korrekt ist |
| add\_password() | Fügt einen neuen, verschlüsselten Eintrag zur Passwortdatei hinzu |
| get\_password() | Zeigt alle gespeicherten Passwörter (nach Entschlüsselung) |
| main\_menu() | Zeigt das Hauptmenü und verarbeitet die Benutzereingabe |

# Funktionsbeschreibung

**generate\_password:** Nutzt /dev/urandom zur sicheren Zufallszahlengenerierung. Gibt nur Buchstaben, Zahlen und \_ aus.

**set\_master\_password:** Liest zweimal ein Passwort ein, prüft Übereinstimmung, speichert SHA256-Hash in Datei .master\_password.txt.

**check\_master\_password:** Liest Hash aus Datei und vergleicht ihn mit dem Hash des vom Benutzer eingegebenen Passworts.

**add\_password:** Fragt nach Benutzernamen und Passwort (oder generiert eines). Verschlüsselt die Kombination mit openssl und speichert sie in .passwords.txt.

**get\_password:** Liest alle Einträge aus .passwords.txt, entschlüsselt sie mit OpenSSL und gibt Benutzername + Passwort aus.

# Code

#!/bin/bash

# Set the password file

SCRIPT\_DIR="$(dirname "$0")"

PASSWORD\_FILE="$SCRIPT\_DIR/.passwords.txt"

MASTER\_PASSWORD\_FILE="$SCRIPT\_DIR/.master\_password.txt"

# Function to generate a random password

generate\_password() {

local length=${1:-16} # Default length is 16

tr -dc A-Za-z0-9\_ <<< $(head /dev/urandom | tr -dc A-Za-z0-9\_ | head -c ${length})

}

# Function to set the master password

set\_master\_password() {

read -s -p "Enter new master password: " master\_password

echo

read -s -p "Confirm new master password: " confirm\_password

echo

if [ "$master\_password" != "$confirm\_password" ]; then

echo "Passwords do not match. Please try again."

return 1

fi

# Store the master password (hash it first!)

MASTER\_PASSWORD\_HASHED=$(echo -n "$master\_password" | sha256sum | awk '{print $1}')

echo "Master password set."

# Store the hashed password in a file (not recommended for real security)

touch "$MASTER\_PASSWORD\_FILE"

chmod 600 "$MASTER\_PASSWORD\_FILE" # Make it readable only by the user

echo "$MASTER\_PASSWORD\_HASHED" > "$MASTER\_PASSWORD\_FILE"

unset master\_password confirm\_password

return 0

}

# Function to check the master password

check\_master\_password() {

if [ ! -f "$MASTER\_PASSWORD\_FILE" ]; then

echo "Master password not set. Please set it now."

set\_master\_password

return 1

fi

stored\_password=$(cat "$MASTER\_PASSWORD\_FILE")

read -s -p "Enter master password: " master\_password

echo

hashed\_password=$(echo -n "$master\_password" | sha256sum | awk '{print $1}')

if [ "$hashed\_password" == "$stored\_password" ]; then

MASTER\_PASSWORD\_HASHED="$hashed\_password"

return 0 # Correct password

else

echo "Incorrect master password."

return 1 # Incorrect password

fi

}

# Function to add a new password

add\_password() {

read -p "Enter username: " username

# Generate a password or let the user enter one

read -p "Generate password? (y/n) [y]: " generate

if [[ "$generate" == "n" ]]; then

read -s -p "Enter password: " password

echo

else

password=$(generate\_password)

echo "Generated password: $password"

fi

# Encrypt the password entry (very basic "encryption")

encrypted\_entry=$(echo -n "$username:$password" | openssl enc -aes-256-cbc -salt -pass "file:$MASTER\_PASSWORD\_FILE")

# Append the encrypted entry to the password file

echo "$encrypted\_entry" >> "$PASSWORD\_FILE"

echo "Password added for $username."

unset username password encrypted\_entry

}

# Function to retrieve a password

get\_password() {

if [ ! -f "$PASSWORD\_FILE" ]; then

echo "No passwords stored yet."

return 0

fi

echo "Stored Passwords:"

cat "$PASSWORD\_FILE" | while read -r encrypted\_entry; do

decrypted\_entry=$(echo "$encrypted\_entry" | openssl enc -d -aes-256-cbc -salt -pass "file:$MASTER\_PASSWORD\_FILE" 2>/dev/null)

if [ -n "$decrypted\_entry" ]; then

IFS=":" read -r username password <<< "$decrypted\_entry"

echo "Username: $username"

echo "Password: $password"

echo "---"

else

echo "Decryption failed for entry."

fi

done

}

# Main menu

main\_menu() {

while true; do

echo

echo "Password Manager Menu:"

echo "1. Set Master Password"

echo "2. Add Password"

echo "3. Get Password"

echo "4. Exit"

read -p "Enter your choice: " choice

case "$choice" in

1)

set\_master\_password

;;

2)

add\_password

;;

3)

get\_password

;;

4)

echo "Exiting."

exit 0

;;

\*)

echo "Invalid choice. Please try again."

;;

esac

done

}

# Check if master password is set and valid

if [ ! -f "$MASTER\_PASSWORD\_FILE" ]; then

echo "Master password not set. Please set it now."

set\_master\_password

else

if ! check\_master\_password; then

echo "Exiting due to incorrect master password."

exit 1

fi

fi

# Run the main menu

main\_menu

```

# Testing

Wir testen das Script auf dem Linux Rechner von Timur und Janis. Finn hat zum Testen ein Programm heruntergeladen damit er es testen kann. Wir haben die einzelnen Funktionen getestet.

Folgendes haben wir getestet:

**Testszenarien:**

* Start ohne Master-Passwort: Aufforderung zur Erstellung funktioniert
* Passwort hinzufügen (mit/ohne Generierung) funktioniert
* Passwortanzeige funktioniert
* Falsches Master-Passwort wird erkannt und verweigert Zugriff
* Dateien .passwords.txt und .master\_password.txt haben nur Nutzer-Lese-/Schreibrechte

**Grenzfälle:**

* Leeres Passwort: nicht erlaubt
* Abbruch bei falschem Master-Passwort

# Fazit

Dieses Bash-Script zeigt, wie man mit einfachen Mitteln eine sichere Passwortverwaltung realisieren kann. Zwar ersetzt es keine professionelle Lösung wie KeePass oder Bitwarden, eignet sich aber hervorragend für kleine private Zwecke oder als Lernprojekt. Durch die Nutzung von OpenSSL und SHA256 wird ein akzeptables Mass an Sicherheit erreicht, auch wenn die Speicherung im Dateisystem Schwachstellen birgt. Für produktive Nutzung sollte ein zusätzlicher Schutz (z.B. mit GPG oder Hardware-Verschlüsselung) erwogen werden.