

PwE - Password Encrypter

Fabian Schrodi, Timo Felser

December 2, 2019

The screenshot shows a web application window titled "PwE - Protect your passwords". The main heading is "Login to your user account". Below this, there are two input fields: the first is empty with a cursor, and the second is labeled "Master-Password". To the right of these fields is a "Login" button. Below the input fields is a "Show input" button with a checkbox. Further down, there is a link that says "... you don't have a username and password yet?". Below this link is the heading "Create user account" and a "New User" button. At the bottom left, there is a gear icon and a button labeled "Options" with a dropdown arrow.

Contents

1	Introduction	2
2	User Guide	3
3	Encryption Algorithm	8

1 Introduction

Nowadays it is common for every website you visit to create accounts for each individual user. Thus nearly for each service you use, you have to think up a password which in addition should be complex enough to be secure. So taking the simplest possible passwords, like 'password', '123456', etc., are as good as having no password in the first place. But on the other hand, it becomes nearly impossible to remember every account you set up together with all the passwords while it is more than easy to loose track of all the different passwords created in the past.

In this project, we aim to solve this problem with a password manager, making your life easier. We developed an encryption algorithm that enables the user to have access to all entries with a single *master password*, the detailed procedure is depicted in section 3. In other words, all the passwords you have ever created are securely stored and you can access all of them by remembering only one *master password*.

In the following, we present the user-guide for this program is given in section 2 and a more detailed description of the encryption algorithm in section 3.

2 User Guide

To be able to use the program you have to either create a .jar-file from the source code, or directly use the file PwE.jar that is already uploaded. The application must be opened with Oracle Java Runtime, leading to the start-screen shown in Fig. 1. If there exist already an account the user can log in here. The language can be set to English or German, via Options -> Language. For help the user can select Options -> Help, which leads to the user manual shown in Fig. 2. In this view the basic abilities of the program are explained, in some similarity to the description here.

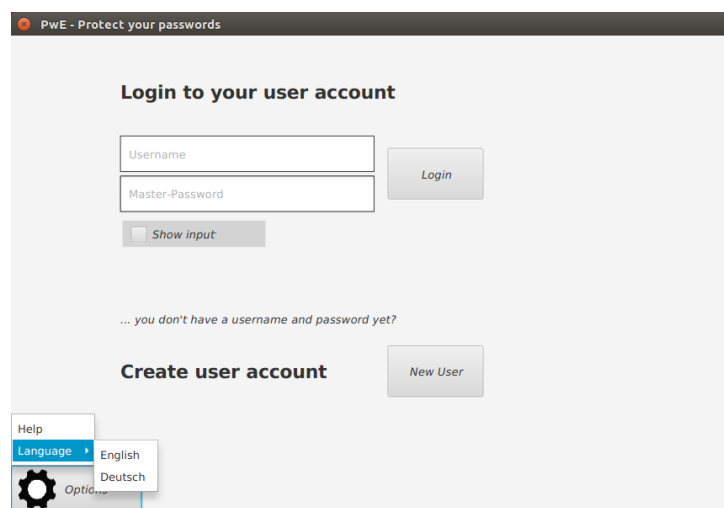


Figure 1: Start-screen.



Figure 2: User manual.

A new account can be created via a click on 'New User' in the start frame, compare Fig. 1. Fig. 3 (left) shows the associated view. Here the user has to choose a single Master-Password, which later enables the access to all other passwords. This Master-Password can either be chosen manually, or generated by the program. In both cases the choice has to be confirmed, where an estimate about the password-strength is shown.

The image shows two parts of the PwE application interface. On the left is the 'PwE - New User-Account' window. It has a title bar with a red close button. Below the title bar is a back arrow button. The window is divided into two sections: 'Username' and 'Password'. The 'Username' section has a text input field containing 'testname'. The 'Password' section has two radio buttons: 'Choose your own password' (selected) and 'Generate your password'. Under 'Choose your own password', there are two text input fields, both containing 'mypassword', and a 'Show input' button with a checkmark icon. Under 'Generate your password', there are two checkboxes: 'Include special characters' and 'Fixed password length', both unchecked. Below these is a 'Choose Length:' label and a dropdown menu. A 'Done' button is located to the right of the password section. On the right is a 'Confirm Master-Password' dialog box. It has a title bar with a red close button. The main text asks 'Are you sure about the password choice?' with a blue question mark icon. Below this, it says 'Your password has a security of 38.04/100'. At the bottom are two buttons: 'Yes' (highlighted in blue) and 'No'.

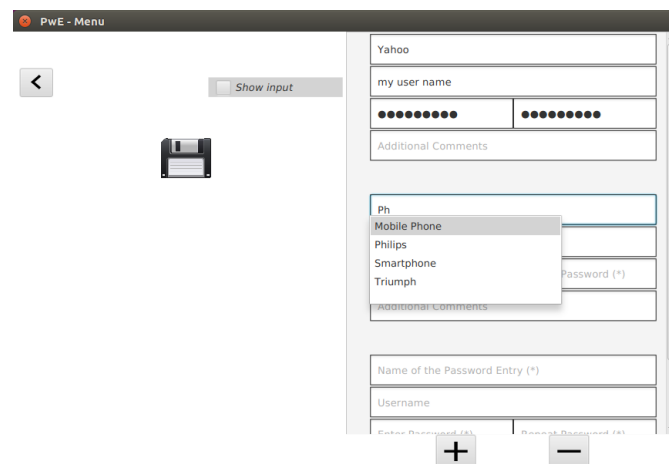
Figure 3: Creating a new user account.

Once a user name and Master-password are picked the user enters the main menu, see Fig. 4. This is the same view as if one uses the login-function in Fig. 1.

The image shows the 'PwE - Menu' window. It has a title bar with a red close button. Below the title bar is a 'Logout' button with a back arrow icon. Below the 'Logout' button are four buttons: 'Encrypt', 'Decrypt', and 'Test Passwords'. At the bottom is an 'Options' button with a gear icon and a dropdown arrow.

Figure 4: Main menu.

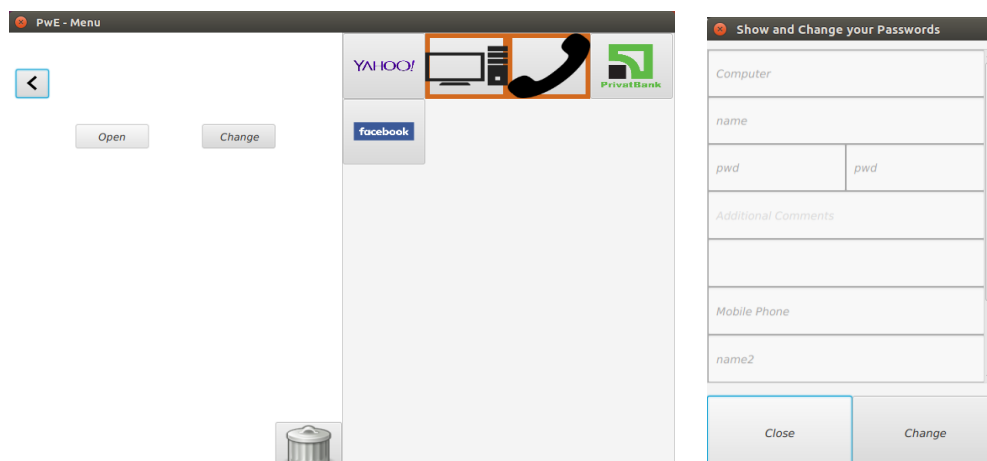
From the main menu a click on 'Encrypt' brings one to the encryption view shown in Fig. 5. Here it is possible to store separate entries for various passwords. Optionally one can give a user name and write some additional comments. We have created a small database for such entries, including common choices such as 'Facebook' or 'YouTube'.



The screenshot shows the 'PwE - Menu' window. On the left, there is a back arrow button and a 'Show input' button. Below these is a floppy disk icon. The main area on the right contains several input fields: 'Yahoo' (with a dropdown menu showing 'Ph', 'Mobile Phone', 'Philips', 'Smartphone', and 'Triumph'), 'my user name', a password field with two masked sections, 'Additional Comments', 'Name of the Password Entry (*)', 'Username', and 'Password (*)'. At the bottom, there are two buttons labeled '+' and '-'.

Figure 5: Encryption view.

All entries stored within a user account are visible in the decrypt frame that we show in Fig. 6. Here every entry is associated with a small logo (if available). The user can show entries, modify them or delete selected ones. We note that once an entry is deleted, it cannot be recovered.



The screenshot shows the 'PwE - Menu' window in the decryption view. On the left, there is a back arrow button, an 'Open' button, and a 'Change' button. Below these is a trash can icon. The main area on the right displays a list of entries with logos: 'Yahoo!' (with a computer icon), 'facebook' (with a Facebook logo), and 'PrivatBank' (with a PrivatBank logo). At the bottom, there are two buttons labeled 'Close' and 'Change'.

Figure 6: Decryption view.

We included the possibility of checking the strength of own passwords. The associated frame, shown in Figs. 7 and 8 is accessible via a click on 'Test Passwords' in the main menu. One can either evaluate a chosen password or test a randomly generated one. In the latter case the user can specify various options that serve as constraints in the password generation.

PwE - Menu

Evaluation of Password-Strengths

☒ Choose your own input

☐ Test a random password

☐ Include special characters

☐ Include empty space

☐ Include numbers

☐ Fixed length

Choose length:

✕

Password to evaluate:
asdf

Strength:
14.26/100

Show password strength

Figure 7: Testing the strength of own passwords.

PwE - Menu

Evaluation of Password-Strengths

☐ Choose your own input

☒ Test a random password

☒ Include special characters

☐ Include empty space

☒ Include numbers

☒ Fixed length

Choose length:

✕

Password to evaluate:
!2E.iY_Zgs

Strength:
100.00/100

Show password strength

Figure 8: Generate passwords.

For storing user accounts and password entires therein we use an external .json file containing all necessary information. An example of such a file is shown in Fig. 9. Although we are confident about the security of all entries, see section 3, this way of storing the data can be a potential flaw. This is due to the possibility of a third party removing or modifying the .json file, which will lead to a loss of all current data.



```
{
  "head": {
    "created": "1570253894014",
    "version": "1.1",
    "user": [
      {
        "name": "testname",
        "numberOfPwds": 5,
        "masterPw": "cc82782e6a60ffa81ba0e1d5e91a1d85dbf535184e95f13377b74bd1:5785a96d7576583fed7e670173eb4ca7c6f5b32bbf4bd392bf43a3df",
        "passwords": [
          {
            "associate": "Yahoo",
            "username": "my user name",
            "encryptUsername": false,
            "masterSalt": "5785a96d7576583fed7e670173eb4ca7c6f5b32bbf4bd392bf43a3df",
            "pwd": "8fcbab05054fa837c9dc0a0a3fcb979d872723345c9c2865cc9fc9fb:459b68abf24a970d96f447af63214673b0ac35b4719c6e538d0db377:274e27ae7cb774bb8398e37cbb016347c900478d13282ec52c448eee",
            "comment": ""
          },
          {
            "associate": "Computer",
            "username": "name",
            "encryptUsername": false,
            "masterSalt": "5785a96d7576583fed7e670173eb4ca7c6f5b32bbf4bd392bf43a3df",
            "pwd": "da528591c44eef0f283031064cbfa49360137225c3fb9d609497828f:074be92629a745b865cebdfa01c477a17878a19126544a9821130021:b01131ad13d6c93b5909db54d248f5cd9e14dbbd5d1ce9373de05f74",
            "comment": ""
          },
          {
            "associate": "Mobile Phone",
            "username": "name2",
            "encryptUsername": false,
            "masterSalt": "5785a96d7576583fed7e670173eb4ca7c6f5b32bbf4bd392bf43a3df",
            "pwd": "b3939083986a41fd50324427db8673204df6a2eb1a25d364e581ffc7:3a0401e7de2f928cde773705107a59a37007ec93323fc0468f816eff:1b204ab98caf6c8f28ebc1b5f4a9530527ff7b1e7fc88a2a364600c7",
            "comment": ""
          },
          {
            "associate": "Privat Bank",
            "username": "name3",
            "encryptUsername": false,
            "masterSalt": "5785a96d7576583fed7e670173eb4ca7c6f5b32bbf4bd392bf43a3df",
            "pwd": "7bc45e29dacf586275a1b87eebef1446fba1c96021eb5b:083ad5ffa27cfe79b6157141ee871d4b893c0846447260206b9a86f3:bf81472d5809fc12401cf7191904fb9396ea30fae8a271de3af9be89",
            "comment": "some comment"
          },
          {
            "associate": "Facebook",
            "username": "fbname",
            "encryptUsername": false,
            "masterSalt": "5785a96d7576583fed7e670173eb4ca7c6f5b32bbf4bd392bf43a3df",
            "pwd": "42d2ba3352c16d2fd779a7ad3864fd76e44f8d9a4f92eef79da3f660:90c06b0690310174bcffae952be3ff8df28194ea85efe1072ef876ef:8ae274793f829b79b5c3164f41e67c188c13e70bfbeec3751f8cb67b",
            "comment": ""
          }
        ],
        "comment": ""
      }
    ]
  }
}
```

Figure 9: Example of a .json file generated by the application.

3 Encryption Algorithm

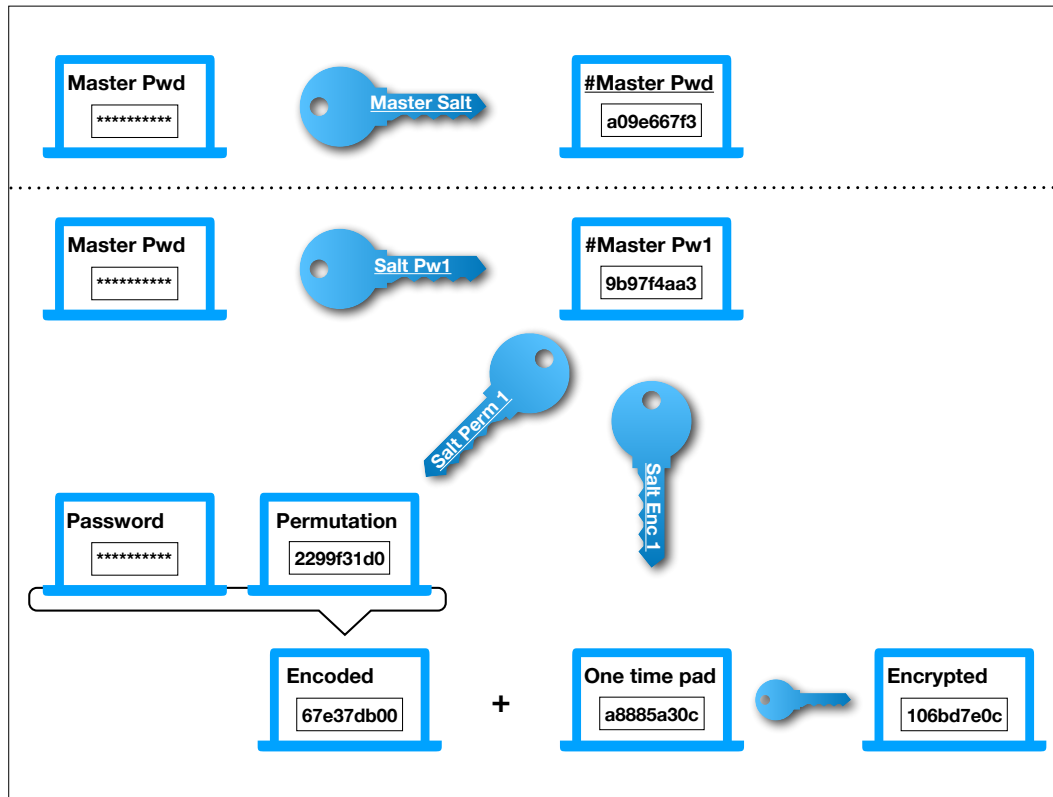


Figure 10: Encryption algorithm for a password. All underlined instances are stored in the program (or after termination in the JSON file).

Our encryption algorithm encodes every account password using the master password chosen by the user. This generic algorithm is illustrated in Fig. 10. Thus, before encrypting and storing an account password, the user has to choose a master password first.

Master password This master password becomes hashed with a random salt. Once the master password is initialised, we store the hashed master password `#Master Pwd` together with the random salt `Master Salt` in the program (or after termination in the corresponding JSON file). In this way, we can only validate the master password when given from the user, but not reconstruct it from the instances we stored. As an example, the stored instances for validating the master password might look like

```
Master Hash   '4f60350533d9ea08b8885d27ee339e5f19c78d11b466518a5c1ceaad4fdea714'
Master Salt   'ccaa30b30a864859a11975af98fd3d25c7e244b619f22e20'
```


In this way, we always can verify whether a given password is your master password without explicitly storing your master password. For the hashing, we use a PBKDF2 with HMAC-SHA as its pseudo-random function which is one of the most secure iterative procedures for hashing passwords.

Encoding new Password After setting the master password, we can add an account for which we aim to encrypt the password (and if desired the username as well). For the encryption of the new password, the master password will be asked for. When inserting the master password, we hash it twice, once with the known **Master Salt**, and once with a new random salt **Salt Pw** obtaining a new hash **#Master Pw1**. The former is used to validate the correctness of the inserted master password while the latter is used to encrypt the new password. The new hashed **#Master Pw** is hashed once again with two random salts **Salt Perm** and **Salt Enc** resulting in two new hashes respectively, the **Permutation** and the **One time pad**.

The new password to be stored becomes encoded in hexadecimal representation and embedded into the created **Permutation** hash. Thereby, the **Permutation** gives the positions for the password characters at which they become embedded. Thus, the first two hexadecimal numbers translate to the position for the first character of the password, the second hexadecimal pairing to the position of the second character, and so on. Thus the **Permutation** determines the permutation for the characters of the password with respect to the hash-size.

After encoding the password into the **Permutation**, the second hash originating from the master password is applied as an on-time-pad. The resulting hexadecimal String is stored (together with the three salts **Salt Pw**, **Salt Perm** and **Salt Enc**) and contains all information about the set password. This password, however, is encrypted and only accessible if the valid master password is given. Thus from the stored instances itself, it is impossible to reconstruct the original password without the given master password. As an example, the stored instances for a password might look like

Encrypted Password	'5de3f098a4d5a4936056bf466100136457f56b21e07f451376a5ec3cd8402756'
Salt Pw	'a9596943036642fddc9722bbafefd2d41863c87449ad43b'
Salt Perm	'c07a2ba4428ad4cf187c3aa2d0cd4112b0bfa77d4aae6ac1'
Salt Enc	'fc2dc59b2a9f8e10f87a8f776b6c3910c73cfdeb696fb64d'

Decoding/Changing a Password In order to decode an encrypted password, the user has to insert the master password. After the validation of the master, the above-described encryption process can be reversed straight forward. With the stored salts we can obtain the one-time-pad used and encrypt the encoded password. Afterward, with the **Permutation** hash we can trace all the positions of the password permutation and reconstruct the password. This process only works when the correct master

password is inserted and fails elsewhere. To change or after viewing a password entry, the encryption process should be repeated with new random salts to obtain a new random one-time-pad and thereby to keep the encryption completely secure.