# Assignment #3 - Password wallet in Go

**Due** Dec 7 by 11:59pm **Points** 200

In this assignment you will build an encrypted password wallet program. The user will enter and modify passwords. Follow the following steps to make this program. Note that this assignment assumes you completed assignment #2 successfully and understand how the go language and workspace operations.

1. Login to your virtual machine. From your virtual machine, download the starter source code provided for this assignment. To do this, create the directories, download the starter file from canvas, and rename it. The file is named swallet443-starter.go.

```
% cd ~/go/src
% mkdir -p cmpsc443/swallet443
% cd cmpsc443/swallet443
% cp ~/Downloads/swallet443-starter.go
% mv swallet443-starter.go swallet443.go
```

2. You also need to install a few things to get this assignment working. Use the commands mentioned in the previous assignment to do the installs as you find you need additional packages.
3. You are to design and implement a password wallet application tool whose specification is given by the following help output:

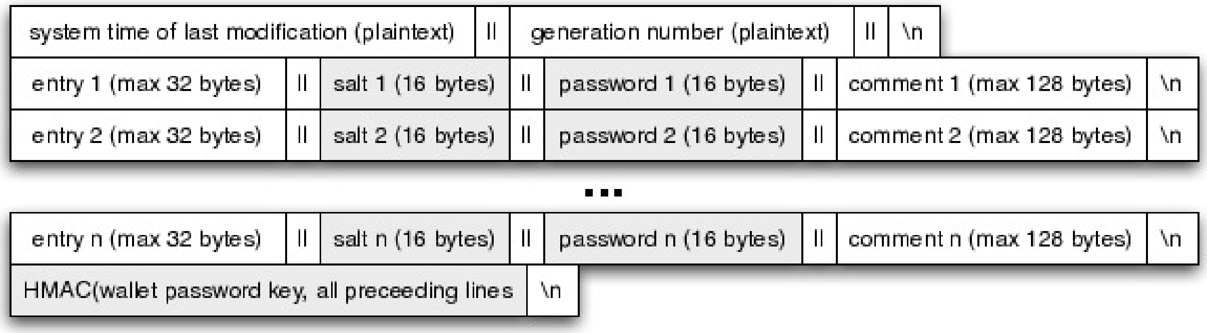
```
USAGE: swallet [-h] [-v]  [create|add|del|show|chpw|reset|list]

where:
    -h - help mode (display this message)
    -v - verbose output

     - wallet file to manage
    [create|add|del|show|chpw] - is a command to execute, where

     create - create a new wallet file
     add - adds a password to the wallet
     del - deletes a password from the wallet
     show - show a password in the wallet
     chpw - changes the password for an entry in the wallet
     reset - changes the password for the wallet
     list - list the entries in the wallet (without passwords)
```

Program notes:

- The wallet is secured through the use of a single master password. It should be initially supplied and confirmed by the user entering it twice and comparing the results during the **create** operation. All HMACs and encryption of data will be performed using this password.
- The user should be prompted to enter the password before performing any action on a previously created wallet. The program should read the wallet file and check the HMAC using the entered

password. If the HMAC does not validate using the entered password, then the program should abort with an error saying bad password.

- The wallet file should confirm to the following schema (all greyed fields should be base 64 encoded):



| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| system time of last modification (plaintext) | \|\| | generation number (plaintext) | \|\| | \n | | | | |
| entry 1 (max 32 bytes) | \|\| | salt 1 (16 bytes) | \|\| | password 1 (16 bytes) | \|\| | comment 1 (max 128 bytes) | \n | |
| entry 2 (max 32 bytes) | \|\| | salt 2 (16 bytes) | \|\| | password 2 (16 bytes) | \|\| | comment 2 (max 128 bytes) | \n | |
| ... | | | | | | | | |
| entry n (max 32 bytes) | \|\| | salt n (16 bytes) | \|\| | password n (16 bytes) | \|\| | comment n (max 128 bytes) | \n | |
| HMAC(wallet password key, all preceeding lines | \n | | | | | | | |

- You should used "encoding/base64" package for encoding and the "crypto" package for all cryptographic functions (hashing, encrypting, etc.).
- The wallet password ($w_k$) is converted into a 128-bit AES key by taking the top most 128 bits (16 bytes) of a SHA1 hash of the password, e.g., $w_k$ = trunc(16,SHA1(password)).
- All passwords to be included in the wallet should be encrypted using 128-bit AES. To encrypt, the password should be left-padded with a unique salt value and right-padded out to 32 bytes with null characters (for example, if the password was 10 characters long, you should pad 22 null (char(0x0)) on the right. More specifically, the password should look like AES($w_k$,$salt_n$|pwd_n|0x0|0x0...).
- The wallet itself must only contain printable ASCII characters. Thus, all salts and encrypted passwords must be base-64 encoded in the file.
- All input should use the some text based UI library. When input is required, the terminal should blank and each input datum should be entered, with appropriate prompt, one per line, starting from the first line. For a sampling of UIs, see **Go language text UIs** (https://appliedgo.net/tui/).
- Password characters should not be echoed to the terminal.
- Each time the wallet is modified, the system time and generation number should be updated. The system time is the local system time (as returned by {time.Now()). The generation number is the number of times the file has been modified. This number should begin at 1 and incremented by 1 for each modification. The wallet file will print these value as in plaintext (human readable text) on the first line.
- The fields of the file are separated using a pair of pipe symbols (ASCII 0x7c), e.g., "||".
- The HMAC for the file should appear as the last line of the file and be base 64 encoded.
- Honors option (and extra credit): Create an option for the wallet program that allows merges. This option should take two versions of the same wallet and combine them in a single file. Use the following rules:
  a. If an entry occurs in one wallet but not the other, it should appear in the output wallet.
  b. If an entry occurs in both but contains different information, the version from the newer wallet should be used. The user should be prompted to make sure this selection is correct.

## To turn in:

You are to turn in the code file your assignment on Canvas. This is a collaborative assignment so each member of a team should submit the exact same file (swallet443.go). The rules of the collaboration should be that each team consists of 3-4 members and you work on the program together as a collaboration. **MAKE SURE** you fill in the top section of the program with all the names of the people on the team.

**Note**: Like all assignments in this class you are prohibited from copying any content from the Internet or getting help from anyone outside of your group. Consulting online sources is acceptable, but under no circumstances should anything be copied. Failure to abide by this requirement will result dismissal from the class as described in our course syllabus. You are explicitly allowed to collaborate within your groups.