POLITECNICO
MILANO 1863

Password Manager

Project for the courses Advanced Operating Systems and Embedded Systems

Authors:

Rosevinay Yelluri
Arlind Rufi
Jan Fischer

Contents

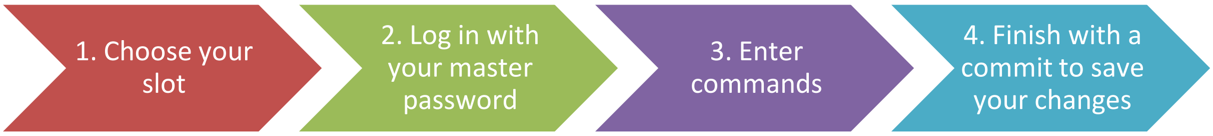
1 Specification

The goal of the project is to implement a password manager as firmware for the STM32F407 board while using miosix as operating system.

The program should provide a command line interface through the serial port using appropriate library functions (e.g. printf and scanf) and allow inputting and retrieving passwords that will need to be stored encrypted on the internal flash. Functions for retrieving passwords should offer a possibility to show all stored passwords with their respective user/website name and in addition a search for a website / user name should be possible.

The driver for the internal flash has been written in a previous project by other students, and is available on the following git repository: https://github.com/goofy91/FileSystem

2 User Manual



1. Choose your slot

2. Log in with your master password

3. Enter commands

4. Finish with a commit to save your changes

Overview how to use the Password Manager

1. There are 4 slots available, simply enter 1, 2, 3 or 4 to choose one. Remember your slot number. You should always use the same to see the passwords that you stored. Other users can use the other slots. One slots offers space for 510 passwords.
2. Make sure to remember your master password and choose a secure one. It allows access to all your stored passwords!
3. Here you can simply enter commands on the console. It is enough to put the first letter of a command. So be sure that you enter the right. Only the first letter is relevant for the program! If you are unsure use the help command to see the names of all commands. These are the available commands:

    - **h(elp)** – this command prints the list of all available commands with a short explanation
    - **a(dd) -** add gives you the chance to add a new website with a password. After inserting this command you will be asked to insert the website and the password for the website.
    - **r(emove) -** this command lets you remove an existing website and its password. You will be asked to insert the website you want to remove. Make sure to write the name exactly as saved. If you are unsure you can use printAll to see the website name.
    - **s(earch) –** with this you can search for a specific website in the data. You will be asked to insert the website of which you want to know the password. Make sure to write the name exactly as saved. If you are unsure you can use printAll to see the website name.
    - **p(rintAll) -** this command gives you the chance to print all the data.
    - **m(asterPassword) –** use this to change the master password with which you login (it will also be used to encrypt your data). Before changing you have to enter your old password.

- **w(ebsitePassword) -** to change the password of an existing website you can use this command. You will be asked to insert the website for which you want to change the password as well as to insert the new password.
- **c(ommit) -** this command allows you to save all the changes you have made. As you see in the diagram above this command should be used in the end of your work. The reason is to protect your flash. Every time commit is used it is erased and will lower the life time of it (usual for flash memory). So it is best to use it just before using exit. If you don't want to save use exit or quit command instead. The commit command writes your data to the flash. Be patient! Encryption and storing on the flash may take up to some seconds. If it is the first time you are using the Password Manager on this slot you will be asked to create the master password. Make sure to remember it.
- **d(elete) -** this command deletes all your data on the flash. This cannot be undone. So be sure that you really want to loose all stored data. Other slots are not affected by it. You can then restart and use the slot on which you used delete like a new one. Since this is a delicate command your identity will be checked again before executing by asking for your master password. The exit command is executed automatically after deleting. If you executed the command accidentally just type no or enter a wrong password.
- **e(xit)/q(uit) –** when you have finished (and have commited) you can quit the program with one of these commands. If you have made changes that you didn't commit you will be asked whether you want to discard them. Type no and use the commit command to save. Type yes to discard all changes you have done in this session (including the change of the master password).

If you exit you will be asked if you want to log in to another slot. Type restart (or anything else) to start the Password Manager again or just press enter to shutdown.

3 Documentation of the implementation

3.1 The user interface

3.1.1 General Implementation

This part of the program is responsible to call all the functions of the password manager in the right order, for interacting with the user and depending on the users input calling the right functions.

In the beginning the Password Manager is initialized by calling the loadData function which will load flash data and set various attributes of the class. To store the commands and other inputs of the user a string is allocated (PASSWORDLENGTH bytes, to store websites and passwords as well). If valid data was loaded the user has to enter the master password. It is used to generate a key to decrypt the data (call of createKey and decrypt function). Goto is used here as a simple construct to reask the password. Afterwards a simple do while is used to allow the user to enter commands. The input is checked with else ifs.

The commands that the user can input are the following (only the first char of the user input is considered):

- **h(elp)** – this command simply uses printf to print the list of all the available commands with a simple explanation next to them.
- **a(dd) -** this command gives the chance to the user to add a new website password tuple. After inserting this command the user will be asked to insert the website and the password for the website. The function addPassword(website,password) is then called having as parameters the users input.
- **r(emove) -** this command gives the chance to the user to remove an existing website password tuple. After inserting this command the user will be asked to insert the website he wants to remove. The function remove(website) is then called having as parameter the website inserted from the user. If this function return false a message is printed telling the user that the website he wrote is not part of the data.
- **s(earch) -**this command gives the chance to the user to search for a specific website in the data. After inserting this command the user will be asked to insert the website he wants to search for. The function searchPassword(website) is then called having as parameter the users input. If this function returns false a message of "not found" is printed.

- **p(rintAll) -** this command gives the chance to the user to print all the data. After the user inserts this command the function printAll() is called.
- **m(asterPassword) -** this command gives the chance to the user to change the master password with which to encrypt and decrypt data. After inserting this command the function changeMasterPassword() is called.
- **w(ebsitePassword) -** this command gives the chance to the user to change the password of an existing website in the data. After inserting this command the user is asked to insert the website for which he wants to change the password. The function changePassword(website) is then called with the users input as parameter.
- **c(ommit) -** this command gives the chance to the user to save all the changes, he has made, in the flash. After inserting this command if it is the first time writing on the flash the user will be asked to insert the password with which to create the key, by calling createKey(), to encrypt the data. The encrypt() and then storeData() functions are then called. In the end the changed attribute is put to false. The commit command will do nothing at all when no changes have been made to protect the flash (no unnecessary erase).
- **d(elete) -** this command gives the chance to the user to delete all his data in the flash. Since this is a delicate command the identity of the user is checked again before executing this command. After verifying the identity the function delete is called.
- **e(xit)/q(uit) -** this command gives the user the chance to end his work and quit the program .If there have been changes during the users work the user will be asked if he wants to commit before exiting or lose all his work.

3.1.2 Functions to support the UI

These are some other functions that help in achieving some of the requisites of the project.

char * addCharacters(char * input): - adds 255 as a Character to an input of size less then 32 and makes it a 32 byte char *. (Char)255 Was chosen because in binary from ascii code is all 1 so as to change the less number possible the flash cells.

void addPassword(char * website,char * password): - adds a WPTuple ,created by the inputs website and password ,in the array of WPTuples and increments numOfPass

char * transformArray(char input[32]): - gets as an input a char array of 32 bytes and removes the (char)255 added by the addCharacters . To be used when you want to print the website or a password .

void printAll(): - function to print all the data contained in the array of WPTuples.

bool searchPassword(char * website): - function to perform queries. Take as an input the name of a website .If this website is in the array of WPTuples prints its corresponding password and then returns true,if it is not return false.

bool remove(char * website): - function to remove from the array of WPTuples an tuple containing the website given as input. If this website exist in the array removes it , decreases numOfPass and returns true, else returns false.

void changePassword(char * website): - function to change the password of the given website as an input. If the website exists in the array of WPTuple it asks the user for the new password ,else tells the user that the website does not exist.

int getPosition(char * input): - function that gives the position of the website as input in the array of WPTuples. Returns this position.

bool deleteData(): - function that erases the data of the user that calls this function from the flash. Returns true if there are no problems during erase.

void scanPassword(char* memory): - function that is used to scan the master password from the console. It will be put in memory, so there is no explicit return.

3.2 Writing to the flash

3.2.1 Flash layout

The data by the Password Manager is saved on the flash. It must be considered that the executable code (namely the operating system) is stored there as well. So one must be careful not to interfere with the data of the operating system. The Password Manager uses two functions to write (storeData) and read (loadData) the data from the flash. Both functions don't use parameters but the attributes of the PasswordManager class. For accessing the flash the memory layout described in table 1 is used. Here the standard start address is used as an example but it is valid for any other address with the corresponding offsets.

Address	Content	Comment
0x080F8000	'P'	Identification string
0x080F8001	'W'	"
0x080F8002	'M'	"
0x080F8003	0x00	"
0x080F8004	numOfPass	Short, which means that it occupies two bytes in the memory
0x080F8006	encryptedData	From here the flash contains WPTuples and the checksum of the password manager, which means all the user's stored password data
0x080F8007	"	Note: all this data is encrypted
...	...	...
EndOfData(= EOD)	0xFF	EOD = 0x080F8006+numOfPass*64+16 Because sizeof(WPTuple) = 64 and sizeof(checksum) = 16
Afterwards	0xFF	Area reserved for new passwords
...	0xFF	"
0x080FFFFF	0xFF	"

Table 1: Flash memory layout

For identification a string with content “PWM” is written at the beginning. In this way we can be sure that there is valid data in the flash and that there is no coincidentally valid data which doesn't make sense.

3.2.2 The loadData function

Changed Attributes	Notes
firstUse	Set to true when no valid data, otherwise set to false
numOfPass	Set to 0 when no valid data. Otherwise loaded from flash
encryptedData	Only changed when data in flash is valid

This function makes sure that all data is loaded correctly from the flash to the RAM. It will set the firstUse boolean and the numOfPass attribute. If data is available the ecryptedData attribute is changed as well. The decryption function is not called by this function.

First the function will load the identifier (the first 4 bytes) from the flash and check if valid data is available (using memcpy and strcmp). As loading data from flash to RAM is always the same, the pattern is only described once for the identifier: We have to get the address of the variable in the RAM as void pointer with a type cast and the address operator (&). However in this special case no &-operator is necessary as a char[] variable contains already the address. A local variable ramAddress is used to store the addresses.
Continuing in the function if valid data is available ("PWM" string found in flash) the data is loaded: first the numOfPass attribute with the same method as for the identifier. Then all the encrypted data, which consists of PASSWORDLENGTH*2*numOfPass+16 Bytes. PASSWORDLENGTH*2 is the size of a WPTuple struct. numOfPass*sizeof(WPTuple) is the space in bytes necessary to store numOfPass passwords. The 16 additional bytes are necessary for the checksum.

To use memcpy correctly it is important to use the correct address in the flash as well. The address attribute is used (which is set once in the constructor), but with the corresponding offset. In the beginning the offset is 0. Afterwards an offset of 4 or 6 is necessary respecting the memory layout in the flash given above.

3.2.3 The storeData function

This function saves the encrypted data to the flash memory using the flash driver class. A boolean variable success is used to verify that the erase call and all write calls have been successful. This boolean is returned by the function (it is the logical AND of all the return values of the called functions). The function writes all the data as described above in the memory layout. It uses the same way for addresses as with memcpy (see loadData) but with char* instead of void*.

3.2.4 Details on address calculation

For using an appropriate address on the flash it is important to note that the OS is stored there as well. The operating system uses the lower addresses on the flash which means it starts at address 0x08000000. It depends on the size of the written firmware i.e. the password manager how much space is occupied.

For the standard address we chose to use the last address in the flash with 32 KiB space afterwards. So it is as far away from the OS as possible. 32 KiB are necessary to store the password data of the password manager. With this it is possible to store up to 510 passwords which should be more than enough for a single person. If not the SlotPasswordManager class offers additional slots to store more passwords. Subsequently the calculation on the standard address is described:

$$32\ KiB = 32 \cdot 2^{10}B = 2^{5} \cdot 2^{10}B = 2^{15}B$$

So we need 2^{15} addresses (one per byte) to have 32 KiB of space. To represent these addresses we need 15 bits. Therefore of the available 20 bit address only 15 bits are necessary. To have the last address we put all bits to 1 apart from the lowest 15. This leads to: 0000 1000 0000 1111 1000 0000 0000 0000.
In hexadecimal representation: 0 8 0 F 8 0 0 0
0x080F8000 (note the 080 as most significant digits are the bus address for the flash).
The address is on the last sector of the used flash which is sector 11.

3.3 Encryption

3.3.1 General

The encryption uses AES working on CBC mode. Before encrypting the checksum of the data is calculated and put on a char array together with all the other data to be encrypted. The checksum is needed after the decryption to check the validity of the data and if the password is correct. The key for the encryption is generated by calculating the hash, using md5 algorithm, of the password given by the user. The user has also the possibility to change the primary password meaning changing the key with which the encryption is done if he knows the old password.

We can divide the functions used to encrypt and decrypt the data in two categories primary functions and secondary helping functions.

The primary functions are the ones that actually encrypt decrypt the data , allow the user to change the masterPassword and also generate the key from the password given by the user. These functions use other functions from aes.c and md5.c libraries which contain the main functions to do CBC encryption/decryption and md5 hash respectfully.

The secondary helping functions that help transform the data structure, the array of WPTuples, used by the program into a char * to be used by the encryption functions in the aes.c and transform the char* given by the decryption into an array of WPTuples to be used by the program.

3.3.2 Specific function implementation

Primary functions:

void Encrypt(): - encrypts the data

- First transform the array of WPTuples into an char * called inputEn.
- Calculate the check sum of this char*,using the mbedtls_md5((const unsigned char *)inputEn,sizeOfInput, checksum) from md5.c, and put it in the variable checksum.(sizeOfInput is the size of the inputEn)
- Create a new char * called toEncrypt and copy here the inputEn and the checksum
- Generate the initialization vector need by the encrypt function from aes.c
- Encrypt the data, and put it in the encryptedData char pointer using the AES128_CBC_encrypt_buffer(encryptedData,toEncrypt,numOfPass*2*PASSWORDLENGTH+16, (const unsigned char*)key,(const unsigned char *)iv), function taken from aes.c

bool Decrypt(char * input): - decrypt the data

- First creates and initializes the initializes the initialization vector the same way as in the encrypt function.
- Decrypts the data given by input and puts the decrypted data in a variable called outputEn, the decryption uses the function from aes.c AES128_CBC_decrypt_buffer(outputEn,input, numOfPass*2*PASSWORDLENGTH+16, (const unsigned char*)key,(const unsigned char *)iv).
- divides the outputEn into two char* one containing the checksum to be checked to see if the password given by the user is correct or not, and another char* toTransform containing the data which if the decryption went correctly is going to be transformed into the array of the WPTuples
- does the hash of the toTransform to compare it to the checksum using mbedtls_md5((const unsigned char *)toTransform,sizeOfInput-16, toCheck) from md5.c toCheck it's a variable where we save the hash of the data gotten from decrypt
- compares toCheck with checksum if they are the same it means the password was correct so it create the WPTuple array and returns true, if they are different it simply return false.

void createKey(char* password): - creates the key to use for encryption and decryption

- as an input has the password given by the user.
- creates the key using the mbedtls_md5((const unsigned char *)password,16, hashedToTransform) hashedToTransform contains the hash of the password and is used to create the Key.
- Since hashedToTransform is 16 bytes and the key needs to be 32 bytes the key is generated by copying hashedToTransform twice in the key variable.
- **changeMasterPassword():-** gives the user the chance to change the master password
- first ask the user for the old password.
- check if the key generated by this password is the same as the key saved in the attribute, if it is ,asks the user for the new password.

Bool checkMasterPassword():- function to check if a given password is the same as masterPassword. To be used when wanting to change the master password or wanting to do delicate operations that need to check the identity of the user like operation delete

- first asks the user to input password
- hashes the users input and compares it to the key that has been used for decryption
- if they are the same returns true

Secondary functions:

char *structToArray(WPTuple input[],int numOfPasswords,int lengthOfWebsite): - transforms an array of WPTuple in a char array

void arrayToStruct(char * input,int numOfPasswords,int lengthOfWebsite): - transforms an char* into an WPTuple and puts it in the passwords attribute

3.4 Multiple user support

A simple class to allow multiple users to store their passwords on the same board has been added. The SlotPasswordManager offers four slots for four different users. It simply uses the PasswordManager class with different addresses. The user has to remember his slot number as well as his master password.

In the current version for each slot it is necessary to use a complete flash sector. This is necessary as the FlashDriver does only provide an erase function for an entire sector. This means putting the data of the PasswordManager to one sector erases all other data stored in the same sector. To save flash space it would be possible to search for other data on the sector by looking for the “PWM” identifier. By this several PasswordManagers’ data could be stored on one sector. However limited RAM size (192KiB) could lead to a problem here: The PasswordManager class alone occupies already more than 64 KiB. Each flash data block is 32 KiB.

The address used for the sector closest to the OS was calculated by the same method as described in *3.2.4 Details on address calculation*. However it does not change anything about the fact that the whole sector cannot be used by the OS (all OS data of the sector will be erased as well when storing the PasswordManager). Therefore it would be the same to use the first address of the sector.

3.5 Further Ideas

The scanPassword function is still a dummy function which simply calls scanf. Actually it should implement an algorithm that hides the input of the user so that the password cannot be seen by others. This functionality can easily be added to the implementation as every time the password is read from the console the scanPassword function is called. So it is only necessary to edit scanPassword.

Moreover to improve security a mechanism should be added that prevents a possible attacker from trying the master password as often as he wants. For this mechanism it is necessary to store data on the flash (if not rebooting would just allow trying again). However this is no problem because the current version is not using the full 32 KiB planned for the Password Manager data (exactly for this reason).

This is the free space available for adding further functionalities: Currently 510 passwords are stored and 16 bytes are necessary for the checksum. However 32KiB are planned. Calculation for PASSWORDLENGTH = 32:

$$\begin{aligned} 32\,KiB - 510\ \cdot 2 \cdot 32\,B + 16\,B &= 2^5\,2^{10}\,B - 510\cdot\ 2^6\,B + 2^4\ \ B \\ &= 2^6(512 - 510)\,B + 2^4\,B =\ 128\,B + 16\,B = 144\,B \end{aligned}$$

Further to improve the multiple user support instead of slots a login with username could be implemented. This would make the login more customisable.

Finally it would still be helpful to edit the linker script to prevent a collision of OS data and stored passwords. So far it has been irrelevant as the application is not yet occupying large space (about 200 KiB). However when continuing the development, it could grow important to prevent errors.

4 Meeting Protocols

1. Meeting on the 28. January 2016

Main topics:

- flash functionality
- implementation details for data structure
- use of available libraries

As we had never done hardware programming with a flash before, we needed a briefing on the basic functionality of the flash. Before we had tried to implement a test program to write on the flash, but didn't know about the necessity to erase it beforehand. Briefly to use flash memory two functions are necessary: write and erase. However the write function can only put values to 0 not to 1. Whereas the erase function does not clear to 0 but to 1. So after erasing every byte of the erased sector contains 0xFF (all ones). The flash driver available only offers an erase of a whole flash sector.
Further it is important to know that erasing the flash destroys it over time; so erase should only be used when necessary. For our project we can reduce the number of used erase calls by using a commit function (not erasing the whole sector for every single password but erasing once in the end of a user session).

With the short time available for implementation in mind we agreed on a simple implementation of the necessary data structure. Instead of using a dynamically growing structure a static structure can be used. The basic idea given by our adviser was to use a struct to store websites and passwords as well as other relevant data on the flash. A password could consist of 32 bytes. The data structure may for instance use 32 KiB.

Most C++ library functions are usable in miosix (the corresponding system calls are implemented). This allows a much faster implementation of various parts of the project: The read function (which is not implemented in the flash driver) can make use of the memcpy function. Encrypting does not have to be implemented from scratch but available functions may be used. Further printf and scanf are available for the user interface. Alternatively the string class with cin and cout could be used.

2. Meeting on the 12. February 2016

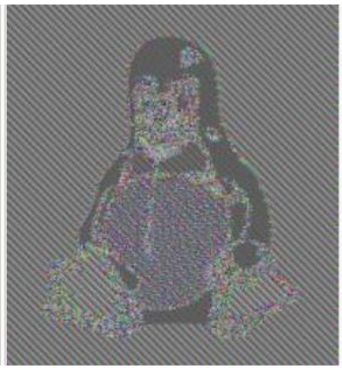
Main topics:

- Flash address
- Encryption
- Optimisation idea

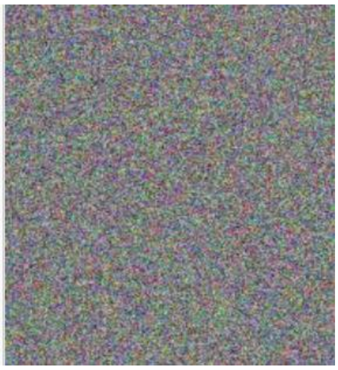
The first address of the flash is 0x08000000 as 080 is the mapping of the flash on the bus. So the lower part is the normal flash address which starts at 00000.
To make sure that the firmware (OS and application) is not overwritten with the data of the PasswordManager it is possible to change the linker script. However the password data won't be as big as to reach the part of the flash where the firmware is stored. But it is important for future references.

For encryption the use of AES with CBC was suggested. This mode was suggested because it is more secure: It hides data patterns in a better way than for example ECB (see pictures below). This is done by XORing each block (sixteen bytes) of plaintext with the previous cyphered block before encrypting.

Original Image



Encrypted using ECB mode



Encrypted using CBC

Source of images: https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#CBC (version from 13.03.2016)
https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#/media/File:Tux_ecb.jpg

The master Password should not be stored on the flash for obvious security reasons. Instead a checksum is stored encrypted with the other data. The checksum is the hash of the valid unencrypted data. To check if the entered password is correct the hash of the decrypted data is compared with the saved checksum.

Our proposed optimisation which was to reduce the number of erases to protect the flash by doing erases only when it is really necessary will not be possible. The idea was to erase only at the first time and when passwords were changed. This would be possible when all data stays the same and we just add data. Because of the checksum it doesn't work since it always changes, also when we only add data.

5 Glossary

AES (Advanced Encryption Standard) – Symmetric key encryption algorithm used to encrypt and decrypt the data to achieve the security of data.

CBC (Cipher Block Chaining) – mode of operating of AES in which each block of plaintext is XORed with the previous ciphertext block before being encrypted.

masterPassword- the password used to generate the key to use for encryption and decryption.

MD5– hashing algorithm used to calculate the checksum of the data before encryption that is going to be used to check if password given by user is correct.

OS – Operating System, here miosix is used on the board

Standard address – see standard start address

Standard start address – used to write/read from flash. It is the last address in the flash with 32 KiB of space behind. This way it is as far away from the operating system as possible. The address is 0x080F8000. It is defined in Passwordmanager.h as STANDARD_ADDRESS.

WPTuple – (or WebsitePasswordTuple) is a struct defined in PasswordManager.h. It is used to store a website and the corresponding password.

XOR – logical operation on bits, also called exclusive or and EXOR. Truth table:

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0