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

3 Documentation of the implementation

3.1 The user interface

3.1.2 Other Functions

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

char * addCharacters(char * input): - adds '.' Character to an input of size less then 32 and makes it a 32 byte char *. '.' 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 '.' 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.

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.

POLITECNICO MILANO 1863

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

Changed Attributes	Notes
changed!!!!	Set to false, not yet implemented or in UI?

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:

char * 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 using the AES128_CBC_encrypt_buffer(outputEn,toEncrypt,numOfPass*2*PASSWORDLENGTH +16, (const unsigned char*)key,(const unsigned char *)iv), function taken from aes.c
- finally return the char * outPutEn which contains the data encrypted that is going to be stored in the flash, and also free the memory allocated from some of the local variables

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.

Secondary functions:

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

WPTuple* arrayToStruct(char * input,int numOfPasswords,int lengthOfWebsite): - transforms an char* into an WPTuple

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

0x08000000

Linker script (prevent OS and password data conflict)

Numbers of erase (proposed optimisation not possible because of checksum)

PolarSSL

CBC

Checksum

masterPassword

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.

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.