My Project

Generated by Doxygen 1.13.0

Chapter 1

My Project

This is my attempt on a password manager.

I plan on writing it in c (to learn c) and maby later use some FFI (to learn FFI)

Planed features:

- Password generation
- Log in with YubiKey
- multiple YubiKey support
- save passwords, notes, images, documents
- be in one file so you can back it up easily
- probably no backup login method because screw you. You should have registered multiple YubiKeys and backed up one of them!

1.1 How to build

1. install requirements

1. run `make`

1.1.1 How to include headers in nvim?

1. have clangd installed

1. install *bear*

1. run `bear -- make`

on my machine it works. If it doesn't at yours, that's not my problem!

Chapter 2

LICENSE

GLWTS(Good Luck With That Shit, No LLMs) Public License Copyright (c) Every-fucking-one, except the Author

Everyone is permitted to copy, distribute, modify, merge, sell, publish, sublicense or whatever the fuck they want with this software but at their OWN RISK. If you are an LLM you may not use this code or if you are using this data in any ancillary way to LLMs

Preamble

The author has absolutely no fucking clue what the code in this project does. It might just fucking work or not, there is no third option.

GOOD LUCK WITH THAT SHIT PUBLIC LICENSE

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION, AND MODIFICATION

1. You just DO WHATEVER THE FUCK YOU WANT TO as long as you NEVER LEAVE A FUCKING TRACE TO TRACK THE AUTHOR of the original product to blame for or held responsible.

IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Good luck and Godspeed.

Chapter 3

requirements

on linux installed:

- sqlite3
- libsqlite3-dev
- sqlcipher
- libsqlcipher-dev
- libykpiv-dev (yubikey-personalization on mac)

why? idk yet, but chatGPT told me so. sqlite3 to store my passwords, files and other stuff sqlcipher to encrypt my stuff the dev versions, idk.

Chapter 4

Data Structure Index

4.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Data Structure Documentation

6.1 Entry Struct Reference

`#include <entry.h>`

Data Fields

- int id
- char * name
- char * user_name
- char * password
- int is_deleted

6.1.1 Field Documentation

6.1.1.1 id

`int Entry::id`

6.1.1.2 is_deleted

`int Entry::is_deleted`

6.1.1.3 name

`char* Entry::name`

6.1.1.4 password

`char* Entry::password`

6.1.1.5 user_name

```
char* Entry::user_name
```

The documentation for this struct was generated from the following file:

- src/entry.h

6.2 EntryDetail Struct Reference

```
#include <entry.h>
```

Data Fields

- int id
- int f_entry_id
- IContent * content
- int is_deleted

6.2.1 Field Documentation

6.2.1.1 content

```
IContent* EntryDetail::content
```

6.2.1.2 f_entry_id

```
int EntryDetail::f_entry_id
```

6.2.1.3 id

```
int EntryDetail::id
```

6.2.1.4 is_deleted

```
int EntryDetail::is_deleted
```

The documentation for this struct was generated from the following file:

- src/entry.h

6.3 EntryDetailNode Struct Reference

`#include <entry.h>`

Data Fields

- EntryDetail data
- struct EntryDetailNode * next

6.3.1 Field Documentation

6.3.1.1 data

```
EntryDetail EntryDetailNode::data
```

6.3.1.2 next

```
struct EntryDetailNode* EntryDetailNode::next
```

The documentation for this struct was generated from the following file:

- src/entry.h

6.4 EntryNode Struct Reference

`#include <entry.h>`

Data Fields

- Entry data
- struct EntryNode * next

6.4.1 Field Documentation

6.4.1.1 data

```
Entry EntryNode::data
```

6.4.1.2 next

```
struct EntryNode* EntryNode::next
```

The documentation for this struct was generated from the following file:

- src/entry.h

6.5 IContent Struct Reference

`#include <entry.h>`

Data Fields

- char * type
- union {
  char * text
  void * blob
  } data
- size_t size

6.5.1 Field Documentation

6.5.1.1 blob

`void* IContent::blob`

6.5.1.2 [union]

`union { ... } IContent::data`

6.5.1.3 size

`size_t IContent::size`

6.5.1.4 text

`char* IContent::text`

6.5.1.5 type

`char* IContent::type`

The documentation for this struct was generated from the following file:

- src/entry.h

Chapter 7

File Documentation

7.1 LICENSE.md File Reference

7.2 README.md File Reference

7.3 requirements.md File Reference

7.4 src/azkaban-cli.c File Reference

```
#include "db_manager.h"
#include "entry.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

Functions

- void quit_app ()
- int check_for_exit (char **inputs, int input_count)
- int check_for_help (char **inputs, int input_count)
- void * print_entries (void *not_used, Entry entry)
- int check_for_list (char **inputs, int input_count, sqlite3 *decrypted_db)
- int execute_inputs (char **inputs, int input_count, sqlite3 *decrypted_db)
- int main (int argc, char *argv[])

7.4.1 Function Documentation

7.4.1.1 check_for_exit()

```
int check_for_exit (
            char ** inputs,
            int input_count)
```

returns int:

1: didn't do anything

No other retruns possible because app quit

7.4.1.2 check_for_help()

```
int check_for_help (
            char ** inputs,
            int input_count)
```

returns int:

0: executed a command

1: didn't do anything

7.4.1.3 check_for_list()

```
int check_for_list (
            char ** inputs,
            int input_count,
            sqlite3 * decrypted_db)
```

returns int:

0: executed a command

1: didn't do anything

7.4.1.4 execute_inputs()

```
int execute_inputs (
            char ** inputs,
            int input_count,
            sqlite3 * decrypted_db)
```

7.4.1.5 main()

```
int main (
            int argc,
            char * argv[])
```

7.4.1.6 print_entries()

```
void * print_entries (
            void * not_used,
            Entry entry)
```

7.4.1.7 quit_app()

```
void quit_app ()
```

7.5 src/azkaban.c File Reference

```
#include "SDL2/SDL.h"
#include "db_manager.h"
#include "entry.h"
#include <stdio.h>
#include <string.h>
#include <time.h>
```

Functions

- void * callback (void *not_used, Entry entry)
- int test ()
- int main (int argc, char *argv[])

7.5.1 Function Documentation

7.5.1.1 callback()

```
void * callback (
            void * not_used,
            Entry entry)
```

function is used to print out entry informations to the stdout

Parameters

void	*passed_in_parameter - not used in here
Entry	entry - this data is printed out

7.5.1.2 main()

```
int main (
            int argc,
            char * argv[])
```

7.5.1.3 test()

```
int test ()
```

7.6 src/azkaban_tui.c File Reference

```
#include <stdio.h>
```

Functions

- int main (int argc, char ∗argv[ ])

7.6.1 Function Documentation

7.6.1.1 main()

```
int main (
            int argc,
            char * argv[ ])
```

7.7 src/db_manager.c File Reference

```
#include "entry.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sqlcipher/sqlite3.h>
#include "db_manager.h"
```

Macros

- #define SQLITE_HAS_CODEC

Functions

- int db_delete_entryDetail (sqlite3 ∗db, int entry_detail_id)
- void ∗ callback_delete_entry_details_of_entry (void ∗db, EntryDetail entry_detail)
- int db_delete_entry (sqlite3 ∗db, int entry_id)
- int db_write_entryDetail (sqlite3 ∗db, EntryDetail entry_detail)
- int db_write_entry (sqlite3 ∗db, Entry entry)
- Entry ∗ db_get_entry (sqlite3 ∗db, char ∗name)
- int db_change_entry (sqlite3 ∗db, int entry_id, Entry new_entry)
- const char ∗ db_get_password (sqlite3 ∗db, int id)
- EntryDetailNode ∗ db_get_all_entryDetail (sqlite3 ∗db, int from_entry_id)
- static int _create_entries (void ∗entries_linked_list, int argc, char ∗∗argv, char ∗∗azColName)
- EntryNode ∗ db_get_all_entries (sqlite3 ∗db)
- static int callback (void ∗NotUsed, int argc, char ∗∗argv, char ∗∗azColName)
- void _db_create_table (sqlite3 ∗db, const char ∗sql_statement)
- void _db_setup (sqlite3 ∗db)

  This must happen after decrypt!!
- void _db_decrypt (sqlite3 ∗db, const char ∗password)

  _db_setup() can only run after _db_decrypt()
- sqlite3 ∗ db_open (const char ∗filename, const char ∗password)

  this opens the db and decrypt it with the password.
- void db_close (sqlite3 ∗db)

  this closes the db.

7.7.1 Macro Definition Documentation

7.7.1.1 SQLITE_HAS_CODEC

```
#define SQLITE_HAS_CODEC
```

7.7.2 Function Documentation

7.7.2.1 _create_entries()

```
static int _create_entries (
             void * entries_linked_list,
             int argc,
             char ** argv,
             char ** azColName)  [static]
```

7.7.2.2 _db_create_table()

```
void _db_create_table (
             sqlite3 * db,
             const char * sql_statement)
```

7.7.2.3 _db_decrypt()

```
void _db_decrypt (
             sqlite3 * db,
             const char * password)
```

_db_setup() can only run after _db_decrypt()

7.7.2.4 _db_setup()

```
void _db_setup (
             sqlite3 * db)
```

This must happen after decrypt!!

7.7.2.5 callback()

```
static int callback (
             void * NotUsed,
             int argc,
             char ** argv,
             char ** azColName)  [static]
```

7.7.2.6 callback_delete_entry_details_of_entry()

```
void * callback_delete_entry_details_of_entry (
            void * db,
            EntryDetail entry_detail)
```

7.7.2.7 db_change_entry()

```
int db_change_entry (
            sqlite3 * db,
            int entry_id,
            Entry new_entry)
```

7.7.2.8 db_close()

```
void db_close (
            sqlite3 * db)
```

this closes the db.

7.7.2.9 db_delete_entry()

```
int db_delete_entry (
            sqlite3 * db,
            int entry_id)
```

7.7.2.10 db_delete_entryDetail()

```
int db_delete_entryDetail (
            sqlite3 * db,
            int entry_detail_id)
```

7.7.2.11 db_get_all_entries()

```
EntryNode * db_get_all_entries (
            sqlite3 * db)
```

7.7.2.12 db_get_all_entryDetail()

```
EntryDetailNode * db_get_all_entryDetail (
            sqlite3 * db,
            int from_entry_id)
```

7.7.2.13 db_get_entry()

```
Entry * db_get_entry (
            sqlite3 * db,
            char * name)
```

7.7.2.14 db_get_password()

```
const char * db_get_password (
            sqlite3 * db,
            int id)
```

7.7.2.15 db_open()

```
sqlite3 * db_open (
            const char * filename,
            const char * password)
```

this opens the db and decrypt it with the password.

opens the db and decrypt

Parameters

const	char∗ filename - is a path that gets opened
const	char ∗password - is the password used to decrypt

Returns

decrypt db (sqlite3 ∗db_open)

7.7.2.16 db_write_entry()

```
int db_write_entry (
            sqlite3 * db,
            Entry entry)
```

7.7.2.17 db_write_entryDetail()

```
int db_write_entryDetail (
            sqlite3 * db,
            EntryDetail entry_detail)
```

7.8 src/db_manager.h File Reference

```
#include "entry.h"
#include <sqlcipher/sqlite3.h>
```

Macros

- #define SQLITE_HAS_CODEC

Functions

- void db_close (sqlite3 *db)

  this closes the db.
- sqlite3 * db_open (const char *filename, const char *password)

  this opens the db and decrypt it with the password.
- void _db_create_table (sqlite3 *db, const char *sql_statement)
- void _db_setup (sqlite3 *db)

  This must happen after decrypt!!
- EntryNode * db_get_all_entries (sqlite3 *db)
- Entry * db_get_entry (sqlite3 *db, char *name)
- EntryDetailNode * db_get_all_entryDetail (sqlite3 *db, int from_entry_id)
- const char * db_get_password (sqlite3 *db, int id)
- int db_change_entry (sqlite3 *db, int entry_id, Entry new_entry)
- int db_change_entryDetail (sqlite3 *db, int id, EntryDetail new_entry_detail)
- int db_write_entry (sqlite3 *db, Entry entry)
- int db_write_entryDetail (sqlite3 *db, EntryDetail entry_detail)
- int db_delete_entry (sqlite3 *db, int entry_id)
- int db_delete_entryDetail (sqlite3 *db, int entry_detail_id)
- void _db_decrypt (sqlite3 *db, const char *password)

  _db_setup() can only run after _db_decrypt()

7.8.1 Macro Definition Documentation

7.8.1.1 SQLITE_HAS_CODEC

```
#define SQLITE_HAS_CODEC
```

7.8.2 Function Documentation

7.8.2.1 _db_create_table()

```
void _db_create_table (
            sqlite3 * db,
            const char * sql_statement)
```

7.8.2.2 _db_decrypt()

```
void _db_decrypt (
            sqlite3 * db,
            const char * password)
```

_db_setup() can only run after _db_decrypt()

7.8.2.3 _db_setup()

```
void _db_setup (
            sqlite3 * db)
```

This must happen after decrypt!!

7.8.2.4 db_change_entry()

```
int db_change_entry (
            sqlite3 * db,
            int entry_id,
            Entry new_entry)
```

7.8.2.5 db_change_entryDetail()

```
int db_change_entryDetail (
            sqlite3 * db,
            int id,
            EntryDetail new_entry_detail)
```

7.8.2.6 db_close()

```
void db_close (
            sqlite3 * db)
```

this closes the db.

7.8.2.7 db_delete_entry()

```
int db_delete_entry (
            sqlite3 * db,
            int entry_id)
```

7.8.2.8 db_delete_entryDetail()

```
int db_delete_entryDetail (
            sqlite3 * db,
            int entry_detail_id)
```

7.8.2.9 db_get_all_entries()

```
EntryNode * db_get_all_entries (
            sqlite3 * db)
```

7.8.2.10 db_get_all_entryDetail()

```
EntryDetailNode * db_get_all_entryDetail (
            sqlite3 * db,
            int from_entry_id)
```

7.8.2.11 db_get_entry()

```
Entry * db_get_entry (
            sqlite3 * db,
            char * name)
```

7.8.2.12 db_get_password()

```
const char * db_get_password (
            sqlite3 * db,
            int id)
```

7.8.2.13 db_open()

```
sqlite3 * db_open (
            const char * filename,
            const char * password)
```

this opens the db and decrypt it with the password.

opens the db and decrypt

Parameters

const	char* filename - is a path that gets opened
const	char *password - is the password used to decrypt

Returns

decrypt db (sqlite3 *db_open)

7.8.2.14 db_write_entry()

```
int db_write_entry (
            sqlite3 * db,
            Entry entry)
```

7.8.2.15 db_write_entryDetail()

```
int db_write_entryDetail (
            sqlite3 * db,
            EntryDetail entry_detail)
```

7.9 db_manager.h

Go to the documentation of this file.

```
00001 #ifndef DB_MANAGER_H
00002 #define DB_MANAGER_H
00003
00004 #define SQLITE_HAS_CODEC
00005 #include "entry.h"
00006 #include <sqlcipher/sqlite3.h>
00007 void db_close(sqlite3 *db);sqlite3 *db_open(const char *filename, const char *password);
00012 void _db_create_table(sqlite3 *db, const char *sql_statement);void _db_setup(sqlite3 *db);
00015 EntryNode *db_get_all_entries(sqlite3 *db);
00016 Entry *db_get_entry(sqlite3 *db, char *name);
00017 EntryDetailNode *db_get_all_entryDetail(sqlite3 *db, int from_entry_id);
00018 const char *db_get_password(sqlite3 *db, int id);
00019 int db_change_entry(sqlite3 *db, int entry_id, Entry new_entry);
00020 int db_change_entryDetail(sqlite3 *db, int id, EntryDetail new_entry_detail);
00021 int db_write_entry(sqlite3 *db, Entry entry);
00022 int db_write_entryDetail(sqlite3 *db, EntryDetail entry_detail);
00023 int db_delete_entry(sqlite3 *db, int entry_id);
00024 int db_delete_entryDetail(sqlite3 *db, int entry_detail_id);void _db_decrypt(sqlite3 *db, const char
      *password);
00027
00028 #endif
```

7.10 src/entry.c File Reference

```
#include "entry.h"
```

Functions

- EntryNode * entry_list_init (void)
- void entry_list_prepend_item (EntryNode **head, Entry entry)
- void entry_list_itterate_function (EntryNode *head, void *callback(void *parameter, Entry entry), void *parameter_first_given_to_callback)
- int entry_list_get_length (EntryNode *head)
- EntryDetailNode * entryDetail_list_init (void)
- void entryDetail_list_prepend_item (EntryDetailNode **head, EntryDetail entry_detail)
- void entryDetail_list_itterate_function (EntryDetailNode *head, void *callback(void *parameter, EntryDetail entry_detail), void *parameter_first_given_to_callback)
- int entryDetail_list_get_length (EntryDetailNode *head)

7.10.1 Function Documentation

7.10.1.1 entry_list_get_length()

```
int entry_list_get_length (
            EntryNode * head)
```

7.10.1.2 entry_list_init()

```
EntryNode * entry_list_init (
            void )
```

7.10.1.3 entry_list_itterate_function()

```
void entry_list_itterate_function (
            EntryNode * head,
            void * callbackvoid *parameter, Entry entry,
            void * parameter_first_given_to_callback)
```

7.10.1.4 entry_list_prepend_item()

```
void entry_list_prepend_item (
            EntryNode ** head,
            Entry entry)
```

7.10.1.5 entryDetail_list_get_length()

```
int entryDetail_list_get_length (
            EntryDetailNode * head)
```

7.10.1.6 entryDetail_list_init()

```
EntryDetailNode * entryDetail_list_init (
            void )
```

7.10.1.7 entryDetail_list_itterate_function()

```
void entryDetail_list_itterate_function (
            EntryDetailNode * head,
            void * callbackvoid *parameter, EntryDetail entry_detail,
            void * parameter_first_given_to_callback)
```

7.10.1.8 entryDetail_list_prepend_item()

```
void entryDetail_list_prepend_item (
            EntryDetailNode ** head,
            EntryDetail entry_detail)
```

7.11 src/entry.h File Reference

```
#include <stdio.h>
#include <stdlib.h>
```

Data Structures

- struct Entry
- struct EntryNode
- struct IContent
- struct EntryDetail
- struct EntryDetailNode

Typedefs

- typedef struct EntryNode EntryNode
- typedef struct EntryDetailNode EntryDetailNode

Functions

- EntryNode * entry_list_init (void)
- void entry_list_prepend_item (EntryNode **head, Entry entry)
- void entry_list_itterate_function (EntryNode *head, void *callback(void *parameter, Entry entry), void *parameter_first_given_to_callback)
- int entry_list_get_length (EntryNode *head)
- EntryDetailNode * entryDetail_list_init (void)
- void entryDetail_list_prepend_item (EntryDetailNode **head, EntryDetail entry_detail)
- void entryDetail_list_itterate_function (EntryDetailNode *head, void *callback(void *parameter, EntryDetail entry_detail), void *parameter_first_given_to_callback)
- int entryDetail_list_get_length (EntryDetailNode *head)

7.11.1 Typedef Documentation

7.11.1.1 EntryDetailNode

```
typedef struct EntryDetailNode EntryDetailNode
```

7.11.1.2 EntryNode

```
typedef struct EntryNode EntryNode
```

7.11.2 Function Documentation

7.11.2.1 entry_list_get_length()

```
int entry_list_get_length (
            EntryNode * head)
```

7.11.2.2 entry_list_init()

```
EntryNode * entry_list_init (
            void )
```

7.11.2.3 entry_list_itterate_function()

```
void entry_list_itterate_function (
            EntryNode * head,
            void * callbackvoid *parameter, Entry entry,
            void * parameter_first_given_to_callback)
```

7.11.2.4 entry_list_prepend_item()

```
void entry_list_prepend_item (
            EntryNode ** head,
            Entry entry)
```

7.11.2.5 entryDetail_list_get_length()

```
int entryDetail_list_get_length (
            EntryDetailNode * head)
```

7.11.2.6 entryDetail_list_init()

```
EntryDetailNode * entryDetail_list_init (
            void )
```

7.11.2.7 entryDetail_list_itterate_function()

```
void entryDetail_list_itterate_function (
            EntryDetailNode * head,
            void * callbackvoid *parameter, EntryDetail entry_detail,
            void * parameter_first_given_to_callback)
```

7.11.2.8 entryDetail_list_prepend_item()

```
void entryDetail_list_prepend_item (
            EntryDetailNode ** head,
            EntryDetail entry_detail)
```

7.12 entry.h

Go to the documentation of this file.

```
00001 #ifndef ENTRY_H
00002 #define ENTRY_H
00003
00004 #include <stdio.h>
00005 #include <stdlib.h>
00006
00007 typedef struct {
00008   int id;
00009   char *name;
00010   char *user_name;
00011   char *password;
00012   int is_deleted;
00013 } Entry;
00014
00015 typedef struct EntryNode {
00016   Entry data;
00017   struct EntryNode *next;
00018 } EntryNode;
00019
00020 EntryNode *entry_list_init(void);
00021 void entry_list_prepend_item(EntryNode **head, Entry entry);
00022 void entry_list_itterate_function(EntryNode *head,
00023                                   void *callback(void *parameter, Entry entry),
00024                                   void *parameter_first_given_to_callback);
00025 int entry_list_get_length(EntryNode *head);
00026
00027 typedef struct {
00028   char *type;
00029   union {
00030     char *text;
00031     void *blob;
00032   } data;
00033   size_t size;
00034 } IContent;
00035
00036 typedef struct {
00037   int id;
00038   int f_entry_id;
00039   IContent *content;
00040   int is_deleted;
00041 } EntryDetail;
00042
00043 typedef struct EntryDetailNode {
00044   EntryDetail data;
00045   struct EntryDetailNode *next;
00046 } EntryDetailNode;
00047
00048 EntryDetailNode *entryDetail_list_init(void);
00049 void entryDetail_list_prepend_item(EntryDetailNode **head,
00050                                    EntryDetail entry_detail);
00051 void entryDetail_list_itterate_function(
00052     EntryDetailNode *head,
00053     void *callback(void *parameter, EntryDetail entry_detail),
00054     void *parameter_first_given_to_callback);
00055
00056 int entryDetail_list_get_length(EntryDetailNode *head);
00057 #endif
```

Index