ISN_RAPPORT/PROJET BITLOCK/18

jules_sanglier ts2

bitlock

bitlock = Project()

bitlock.self.about = "Projet de fin d'année créé entre 2017 et 2018 lors de l'enseignement de spécialité Informatique et Sciences du Numérique en Terminale Scientifique au lycée Charles Péguy à Orléans par **Donatien COMBY** et **Jules SANGLIER**."

bitlock.upload2(« github.com/jsangli3r/Bitlock")
bitlock.self.version = "alpha 13517"
bitlock.initIndex()

Version numérique :

Repo Github : intégralité projet

Sommaire

| Présentation

A. Enjeux et problématique, pourquoi ce projet ?

B. Positionnement du projet, solutions avec objectif équivalent

C. Objectif final

| Application

A. Cahier des charges de l'équipe

B. Pré-requis matériels et logiciels

C. Organisation

D. Fonctionnement concret

| Réalisation / Intégration personnelle

A. Fichier principal C/C++

B. L'interprétation des commandes

C. Gestion des données

D. Contribution / aide software manager

| Bilan / Perspectives / Diffusion

Présentation

A. Enjeux et problématique, pourquoi ce projet ?

On peut partir du constat qu'actuellement en 2018 afin d'accéder à un contenu sécurisé nous sommes dans l'obligation d'utiliser **deux concepts**, un identifiant (public) et une phrase secrète (un mot de passe) que seul le propriétaire est en mesure de connaître. Ce système normalisé de sécurité fait aujourd'hui débat. Au vu des progrès croissants de la puissance de calcul (Moore a eu tort) qu'un individu peut obtenir avec les nouveaux processeurs et les nouveaux procédés cryptographiques de récupération de données une nouvelle contrainte apparaît à l'utilisateur lambda. Celui-ci **ne peut plus utiliser un mot de passe suffisamment simple** comme il fut habitué (c'est à dire une dizaine de caractères qui ont du sens à ses yeux) il y a quelques années car ceux-ci sont **faillibles** dans la mesure où ils sont facilement atteignables pas une attaque de type brute force, c'est à dire un test de toutes les combinaisons possibles via un dictionnaire.

C'est pourquoi nos mots de passes sont désormais la plupart du temps soumis à des règles (des regex) afin de **permettre l'intégrité** de ceux-ci. On peut dénombrer plusieurs facteurs comme la longueur totale ou la difficulté syntaxique voire le bruit de données (données sans aucune représentation réelle). Afin de s'adapter à ces nouveaux concepts, l'utilisateur lambda décide donc d'opter pour la sécurité car il est soucieux des données que son mot de passe protège, il adopte donc la méthode de génération aléatoire de mot de passe. Dès lors, il obtient un mot de passe d'une longueur donnée avec des caractères alphanumériques ainsi que des caractères spéciaux qui permettent l'intégrité théorique des données dans le cas d'une attaque par brute force car le nombre d'essais avant de trouver la combinaison exacte de caractères alphanumériques + caractères spéciaux est très conséquent. Or, M. lambda possède des comptes sur plusieurs sites et possède donc plusieurs mots de passe qui n'ont désormais aucun sens à ses yeux, de plus ils sont longs … Les apprendre par coeur serait une perte de temps et d'énergie.

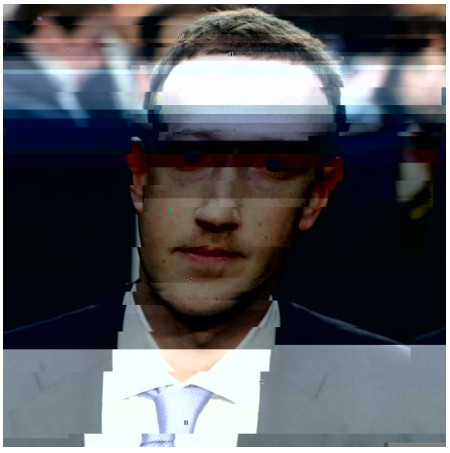
B. Positionnement du projet, solutions avec objectif équivalent

Des solutions ont été créées telles que la sauvegarde des identifiants dans le navigateur utilisé lors des connexions, mais cela pose divers problèmes tels que l'utilisation de votre navigateur par un tiers ou un quelconque logiciel (comme une extension) qui aurait trouvé une **faille critique** dans le navigateur. Il est important de rappeler qu'il existe aussi des sociétés qui ont crées des applications notamment sur smartphone : vos mots de passes sont alors gérés par des tiers, généralement ces applications **ne sont pas open-source** et donc le lieu de sauvegarde des données ne vous est pas connu : dès lors qui peut affirmer que vos **données sont stockées sur votre appareil**... Vous laissez donc votre **confiance en prime de vos données et d'une gratuité qui se paie par le partage de vos données personnelles** (Hi Facebook). Que faire si du jour au lendemain la société décide de suspendre l'application ? Nombreux sont les cas de sociétés informatiques ayant fermées à cause de leur implication dans des sujets sensibles qui entachent leur image et nuisent à leur réputation, on peut prendre le cas récent de **Mt. Gox** (plateforme d'échanges de Bitcoins).

Par ailleurs, il est important de noter qu'un **projet visiblement équivalent**[12] sur le concept : (la sécurité en moins) a été crée, sous la forme d'une clé USB, cependant ce système possède un écran pour récupérer les identifiants ce qui pose des problèmes de confidentialités visuelles.

Il subsiste néanmoins la technique élémentaire du papier crayon scellé dans un coffre, si vous avez accès à votre compte bancaire tous les jours afin de laisser en sécurité vos données (mais vous êtes encore soumis à un tiers). Peu pratique me direz vous...

La meilleure solution vous en conviendrez reste donc d'être totalement indépendant mais propriétaire de nos données et pouvoir les gérer via un système unique qui allie sécurité, vie privé et simplicité d'usage.



'We're not forcing anyone to publicize any information about themselves' – M.Zuckerberg

C. Objectif final

L'objectif majeur de ce projet est donc de maximiser la sécurité et la simplicité d'usage de l'utilisateur vis à vis des identifiants qu'il peut utiliser sur Internet et ailleurs. Et ce, afin qu'il puisse se soucier au minimum de la confirmation de son identité. Nous avons donc décidé de créer un système portable (un périphérique, esclave), qui en étant lié à un autre appareil pourra communiquer via un logiciel sur un ordinateur (maître) afin d'acheminer en toute sécurité les identifiants de manière cryptographique via un port série (USB/TTL). Le tout en liant un système de déverrouillage via un capteur d'empreintes qui permettra un accès personnel et sécurisé.

Par ailleurs, il est important de rappeler que nous sommes tout à fait conscients des limites du projet dans la mesure ou dans quelques années les identifications par mots de passes auront disparus. Nous espérons qu'ils laisseront ainsi place à une identification physiologique. Le but n'est donc pas ici de résoudre le problème de l'identification mais d'apporter une solution sur le court terme[1] afin de rendre l'utilisation des identifiants plus agréable pour l'utilisateur tout en maximisant la sécurité des données qui transitent.

L'objectif majeur est donc de pouvoir déverrouiller un système sécurisé à plusieurs facteurs qui permet de retrouver ses identifiants mais aussi de remplir automatiquement via un process d'autocomplétion des pages de logins (identifiants + mot de passe).

Application

A. Cahier des charges de l'équipe

Le cahier des charges peut se définir sous la forme de contraintes imposées par les valeurs du projet et des créateurs. Nous devons concevoir un objet portable, compact, ayant la capacité de stocker plusieurs dizaines de mots de passe ainsi que des données liées : on pense notamment aux sites web (un indicateur de reconnaissance pour l'utilisateur) ainsi que les emails / pseudos. Ces données doivent être stockées de manière sécurisées et modifiables à tout moment. Cependant, d'un point de vue cryptographique certaines valeurs devront être immuables dès lors qu'elles seront définies par l'utilisateur. On pense à l'utilisation d'un algorithme de cryptographie symétrique, c'est à dire que l'utilisateur posséderai une clé de déchiffrage courte de toutes les données. Afin de renforcer la sécurité, nous rajoutons un facteur d'identification en plus, un moyen de déverrouiller l'appareil par le biais d'un capteur d'empreinte digitale. Dès lors, sans ces deux facteurs il serait impossible d'accéder aux données sauvegardées dans le périphérique externe.

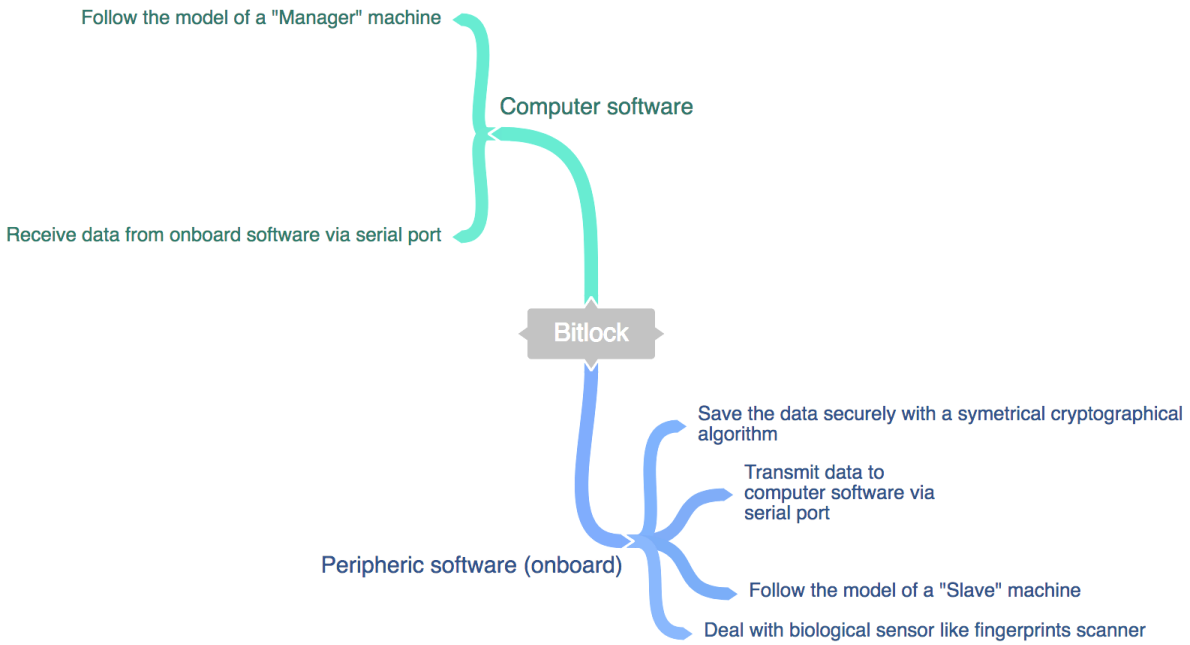


Fig. 1 - Carte mentale du cahier des charges

B. Pré-requis matériels et logiciels

Afin de réaliser à bien ce projet qui lie électronique et programmation nous avons besoin de différents supports matériels et logiciels. Pour ce qui est du périphérique nous avons privilégié pour des questions de facilités l'emploie d'une carte Arduino (Mega, **Fig. 2**) pour le support des données. L'Arduino est un bon choix dans la mesure où il coûte peu cher et qu'il dispose d'une mémoire morte (EEPROM) pour sauvegarder les données (4Ko, suffisant pour le projet) et fonctionne en 5V avec une quarantaine de sorties avec une intensité possède de 40mA chacune. Le programme sur l'Arduino est codé en C/C++ avec l'utilisation de différentes librairies, notamment celle de l'EEPROM (crée par Arduino) et SofwareSerial pour communiquer entre le capteur d'empreinte et l'Arduino.

Nous utilisons donc aussi un capteur d'empreinte biométrique (**Fig. 3**) de chez SparkFun qui a la capacité de sauvegarder une trentaine d'empreintes biométriques et est capable de reconnaître une empreinte à 360° (selon le constructeur…). Ce capteur a la capacité de fonctionner via le port série (TTL) ce qui nous arrange bien pour la communication. Il fonctionne nominalement en 3.3V, nous devons donc abaisser la tension de sortie de l'Arduino qui est de 5V. C'est pourquoi nous utilisons des résistances, 4*330 Ω (toutes ne servent pas à abaisser cette tension particulièrement). Nous utilisons par ailleurs un buzzer (composant résonnant à une fréquence donnée lors d'une stimulation électrique) pour indiquer à l'utilisateur si le doigt posé sur le capteur d'empreinte est valide.

Afin de facilité les différents tests nous avons besoin d'un système sans soudure pour éviter les erreurs, nous avons choisi une breadboard (**Fig. 4**).

L'Arduino quant à lui est piloté par un logiciel (`Manager`) codé en python (pour des raison pratiques et de connaissances). Pour communiquer via le port série, les deux logiciels doivent se comprendre. Nous avons donc établis un protocole d'équivalences logiques entre les deux. Pour fonctionner nominalement, le Manager doit posséder les librairies suivantes :

- `threading` (natif) : afin de faire un multi-thread car on ne peut pas gérer l'interface graphique et la réception perpétuelle de données (par une boucle while) en provenance du port série
- `PyQt`[2] : dérivé du logiciel de Nokia QtCreator cette librairie permet de faire des interfaces au sein de projets python à la place du C++ usuel pour lequel elle fonctionne normalement
- `pyserial`[3] : permettant de s'attacher à un port série et de communiquer avec
- `pyautogui`[4] : permettant d'émuler les actions d'un humain sur un clavier, une souris, très utile pour remplir des formulaires web !
- `pyperclip`[5] : elle permet la manipulation du clipboard de l'ordinateur (utilisé en complément de pyautogui). Tous les liens de ces librairies sont disponibles en Annexe.

Pour programmer en python, j'ai pour ma part opté pour PyCharm[6] de NetBeans qu'on ne présente plus pour tous ces raccourcis et son autocomplétion qui rendent la vie beaucoup plus agréable. En C/C++ pour l'Arduino j'ai choisi d'utiliser l'IDE officiel[7], peu pratique ne gérant pas le version-control-system mais robuste.

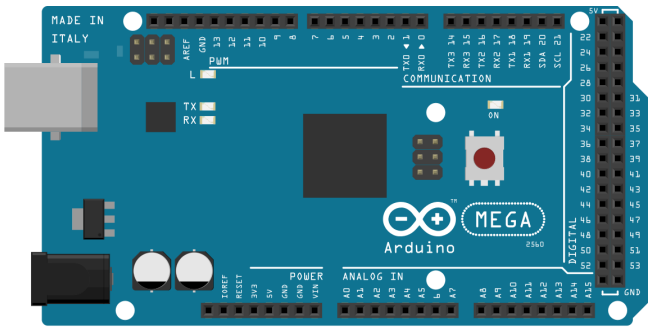


Fig. 2 - Arduino Mega 2560 Rev 3

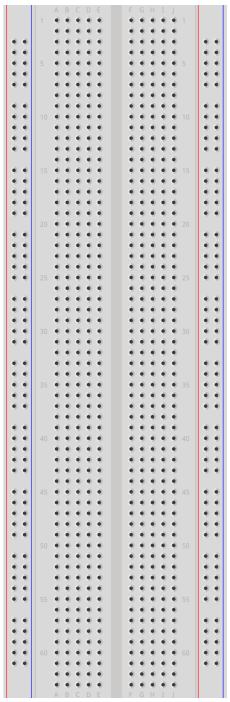


Fig. 3 - Breadboard

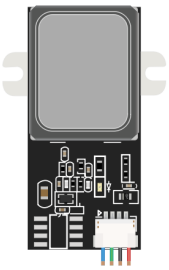


Fig. 4 - GT-511C1R, scanner d'empreintes digitales

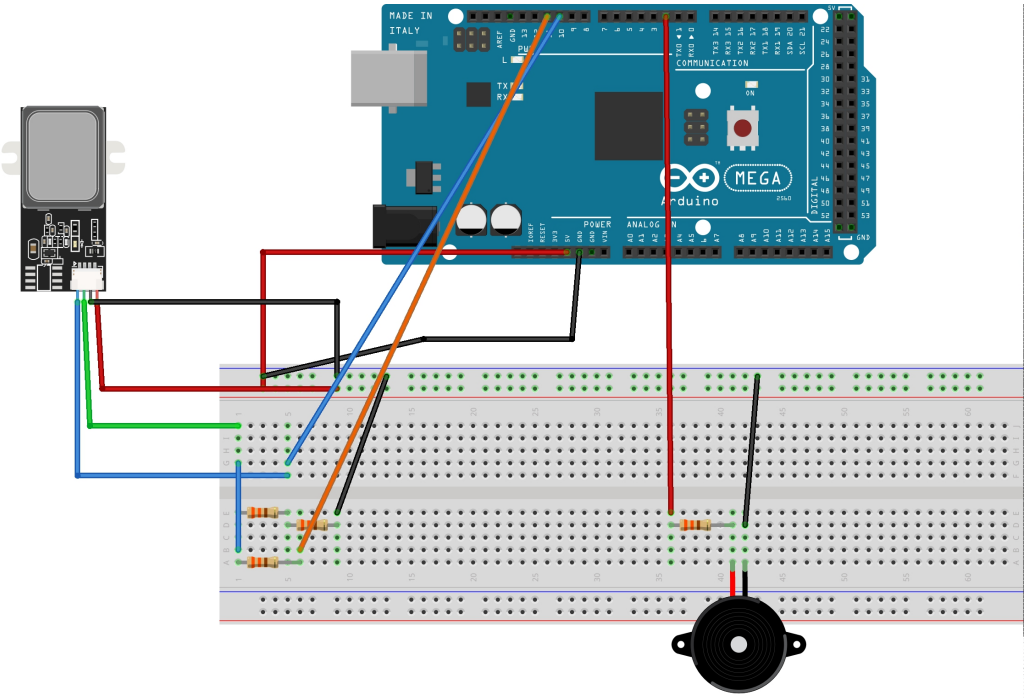


Fig. 5 - Schéma de câblages de l'ensemble du système onboard, schéma électrique disponible en Annexe

C. Organisation

Afin d'être efficaces et productifs nous avons mis en place plusieurs systèmes de gestion et d'organisation du projet, notamment les plateformes Trello[8] : afin d'assigner les tâches, l'organisation, et prévoir les deadlines; Bitbucket[9] : pour travailler de manière collaborative en évitant les collisions avec les versions existantes, grâce à git[10]; ainsi que Slack[11] : pour partager des messages, échanger des fichiers. Les séances au lycée sont avant tout dédiées à faire le point sur l'avancement et répondre au plus grosses problématiques, aux éventuelles idées complémentaires. Ainsi la majeure partie du travail s'effectue chez nous en communiquant avec les outils mentionnés ci-dessus.

Nous sommes partis avec l'idée et la répartition suivante, pour ma part je devrais me charger de toute la partie onboard du projet et Donatien devrait s'occuper de la majorité du logiciel Manager mais une aide ambivalente bien évidemment possible entre les deux logiciels. Nous avons donc pu commencer à proposer des idées sur le fonctionnement grâce à de multiples cartes mentales (et les feutres de M. Leroux 😊). Sur notre tableau Trello toutes les dates clés étaient fixées, nous avons fait en sorte de coder les deux programmes de manière conjointe, tout d'abord nous devions résoudre le problème de la communication entre les deux en partageant un protocole entre les deux logiciels, puis ensuite toutes les fonctionnalités et enfin l'interface de gestion du Manager.

Un date en particulier était définie depuis le début : la deadline qui signifiait que nous devions avoir fini le 13 mai au soir les deux programmes (spoil : ce fut un échec).

D. Fonctionnement concret

Lors de l'établissement des concepts décisifs du fonctionnement de l'ensemble du système, nous avons été confronté à quatre problématiques majeures :

- le protocole de communication par port série, α : pX
- la sauvegarde des données, β : pX
- la sécurité à double facteur, ɣ : pX
- l'autocomplétion des formulaires de logins, δ : pX

Nous allons tout d'abord expliquer brièvement comment nous avons répondu à ces problématiques d'une manière technique, puis nous aborderons donc individuellement chacune d'elles dans cette partie.

α) Le protocole de communication par port série

Le protocole de communication à établir doit respecter des règles qui seront connues entre les deux différents modules à savoir le software onboard et le manager. On a choisi de simplifier au maximum le problème en évitant un mode de communication par

packets (donc seulement sous une forme décimale). Cela aurait conduit à une difficulté de compréhension globale et aurait mis beaucoup trop de temps à être mis en place. Nous avons donc choisi de définir les règles suivantes, chaque commande doit être sous la forme :

$$\Phi + \sum_{k=0}^{n} p_k$$

où : - Φ désigne la commande
- p_k désigne un paramètre k

Entre chaque paramètre (et la commande) il est essentiel afin de les délimiter de placer un caractère de référence; ici il s'agira d'un espace et ce si et seulement si $k \neq n$. Les commandes après interprétation permettent d'exécuter une action définie qui est contenue dans une fonction auxiliaire et individuelle à l'interprétation. Pour différencier les demandes des réponses, nous avons décidé d'ajouter simplement un suffixe à chacune d'elles :

- « `in` » pour les commandes entrantes (par rapport à l'Arduino)
- « `out` » pour les commandes sortantes (par rapport à l'Arduino)

Une commande entrante doit toujours donner lieu à une commande sortante c'est la règle. De plus, nous avons ajouté la possibilité à un seul paramètre p_k d'être une chaine de caractères complexes pouvant supporter les espaces (le caractère de référence de délimitation), c'est à dire que tout ce qui est entre « » pour un paramètre donné pourra produire un paramètre unique : ceci est utile pour envoyer des messages d'informations, d'erreurs (de l'Arduino vers le software manager). La liste des commandes est disponible en annexe et dans le projet. Pour l'interprétation des commandes sur l'Arduino il suffit de monitorer le port série et de router les données vers un interpréter. Sur le software manager, c'est un peu plus compliqué et il faut faire appel à une technique de multi-threading, car le logiciel fonctionne usuellement sur un seul thread cependant nous devons monitorer continuellement les données entrantes par le port série (à l'aide d'une boucle) donc forcément il y a conflit et freeze voire erreur de segmentation mémoire.

β) La sauvegarde des données

Ce que nous définissons ici comme les données a été assimilé à un concept que nous avons nommé `Payload` (littéralement charge utile). Ce concept peut être instancié grâce à trois attributs : le website (le repère contextuel de l'utilisateur), l'identifiant, et le mot de passe. Ces Payload afin d'être sauvegardés en mémoire, c'est à dire dans la mémoire morte de l'Arduino : l'EEPROM de 4Kb doivent être structurés (on explique ce concept un peu plus loin) afin d'être identifiés par un algorithme de recherche (payload pool finder). La mémoire peut être vue comme un tableau à une dimension : à n lignes possédant tous un indice *k* et un contenu en colonne supplémentaire noté *x*. Il est à

Fig. 6 - Représentation conceptuelle de la mémoire

kn	x
k_1	x_1
k_2	x_2
…	…
k_n	x_n

noter que pour stocker des chaines de caractères sous cette forme nous ne pouvons pas les sauvegarder de manière brut sous leur forme complexe ou simplement lettre par lettre, car objectivement l'EEPROM ne sait pas ce qu'est un caractère. Cependant, physiquement elle peut manipuler les représentations décimales (via les électrons). On a donc dû adapter un système international qui permet de convertir des valeur décimales vers des caractères, le tableau ASCII (**Fig. 7**) étant toujours utilisé aujourd'hui malgré qu'il date du début des années 70. Il a prouvé son efficacité et sa portabilité.
Ainsi, en utilisant un tel tableau, on peut sauvegarder tous les caractères usuels sous

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Fig. 7 - Tableau ASCII

forme d'un bit unique à une adresse k_n. Les caractères étant désormais coupés en de multiples morceaux allant de k_n à k_{n+x} nous devons désormais structurer les données. Pour structurer les Payload nous avons fait appel à des « patterns de reconnaissances », c'est-à-dire des valeurs de références dans la mémoire que l'utilisateur ne peut pas utiliser dans un Payload et qui servent donc de bornes structurelles lors de la récupération des caractères via un algorithme de recherche. Ainsi, nous avons définis arbitrairement 4 valeurs structurelles :

- header_start : début d'un Payload (unique), 1
- head_data_start : début de données (*3 / Payload), 2
- head_data_stop : fin de données (*3 / Payload), 3
- header_stop : fin d'un Payload (unique), 4

Désormais, ce concept peut être appliqué à des données quelconques dans la mesure où $0 \leq k_n \leq 4\text{Kb} \Leftrightarrow 0 \leq k_n \leq 4046$ bits et où x_n est une valeur convertie du système de caractère vers une représentation décimale via le tableau ASCII.

Fig. 8 - Payload quelconque (à une seule donnée, normalement *3)

k_n	x_n	Description
0	1	header_start
1	2	head_data_start
2	120	x
3	46	.
4	99	c
5	111	o
6	109	m
7	3	head_data_stop
8	4	header_stop

Ɣ) La sécurité à double facteur

La sécurité est un axe majeur du projet dans la mesure où tout l'interêt repose dessus. Malheureusement, le temps nous a manqué pour introduire ce concept de manière complète, on va donc ici étudier comment théoriquement cela fonctionnerait. Cependant sur deux facteurs, un est partiellement fonctionnel : la reconnaissance d'empreintes digitales. Au départ, pour déverrouiller intégralement le système, il fallait, que les deux conditions soient réunie : empreintes + clé privé (la seule), nous avons établi un système d'état de sécurité qui fonctionne conjointement avec ces deux principes en passant outre la clé privé.

Pour ce qui est de la cryptographie, l'idée initiale fut l'utilisation d'un algorithme de hachage comme BCrypt[13]. Cet algorithme permet une complexité croissante selon le temps de génération du hach (on peut donc faire face à du brute force), il est donc adaptatif en plus d'utiliser un sel (une valeur privée appliquée par dessus les valeurs hachés et des haches multiples… Cela permettrai de hacher les données avec une clé privée sous la forme d'un unique mot de passe simple, ainsi on aurait juste à vérifier via une commande si la valeur entrée est celle contenue dans la mémoire du programme Arduino sous forme d'un String. Or, on peut flasher des programmes mais pas les récupérer à la différence des valeurs de l'EEPROM. Ainsi, pour changer cette clé privée il faudrait flasher de nouveau la totalité du programme. Quand bien même que quelqu'un voudrait flasher un programme et explorer l'EEPROM toutes les valeurs seraient hachées avec cette clé, ainsi l'intégrité de celles-ci seraient préservées. À l'heure actuelle, le projet transmet toutes les données en clair par le port série.

Pour le capteur d'empreintes, il était prévu au départ d'utiliser une combinaison d'empreintes pour déverrouiller le système. Cependant le capteur sauvegarde dans sa propre EEPROM les données d'empreintes, or celle-ci n'est pas modifiable via la librairie, ainsi, faire un système de la sorte aurait demandé de réserver une zone dans l'EEPROM de l'Arduino pour sauvegarder une combinaison. On s'est donc contenté d'un seul doigt pour valider le déverrouillage.

δ) L'autocomplétion des formulaires de logins

L'autocomplétion permet le remplissage d'un champ textuelle automatiquement, ainsi ce concept se révèle fatalement utile pour simplifier l'utilisation d'identifiants, pas besoin de copier-coller hasardeux, le software manager se charge de tout ! Nous avons réduit le problème le plus simplement possible, les données reçues via le port série sur le software manager sont ainsi isolés et via un bouton l'utilisateur lance le processus d'autocomplétion. Ce processus permet au bout d'une pause du thread principal de 10s de copier virtuellement les données et de les coller, en émulant des touches via les librairies pyperclip et pyautogui. Ainsi, on copie l'identifiant, on le colle avec un Ctrl/Command + V, on émule un Tab puis on répète le processus avec le mot de passe et on appuie sur Entrer (on suppose donc ici que le formulaire est sous la forme usuel identifiant / mot de passe). Outre la partie technique, l'utilisateur sélectionne le site web sur lequel il veut se connecter via notre logiciel, il presser le bouton « Output the selected payload » (**Fig. 9**), il a 10 secondes pour se placer sur la page de login dès lors, les identifiants sont saisis automatiquement !

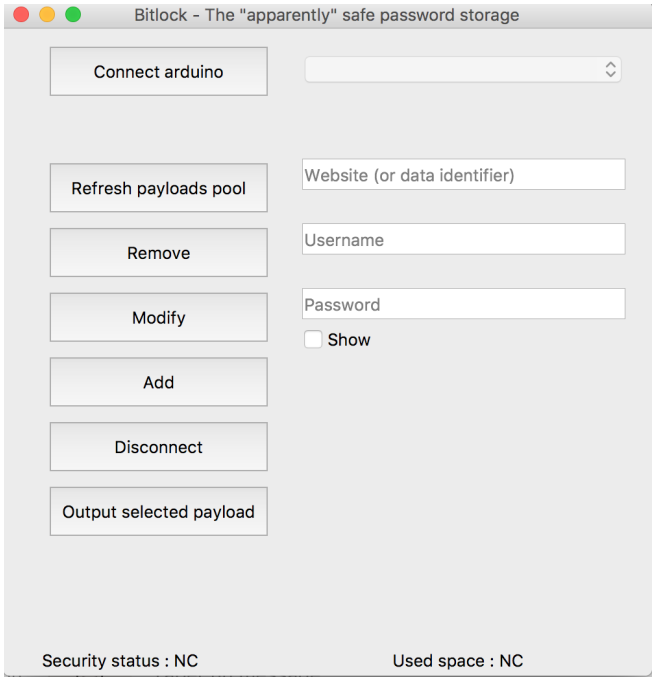


Fig. 9 - Interface du software manager

Réalisation

Le travail le plus conséquent que j'ai effectué est l'écriture totale du programme de l'Arduino : onboard software, qui est en C/C++. Je vais donc présenter ici les différents concepts et paradigmes que j'ai dû suivre pour mener à bien le coeur du projet. J'ai rencontré de nombreux problèmes lors de la réalisation de ce programme malgré des bases solides en programmation, j'ai eu très peu l'occasion de faire du C/C++ auparavant, ainsi au fil des tests et nombreuses lectures j'ai pu atteindre une version fonctionnelle, je pense que la plus grande difficulté fut la gestion des chaînes de caractères qui représentent littéralement un enfer en C lorsque l'on vient de langage comme le PHP. J'évoquerai aussi de manière nuancée ma contribution au software manager puis nous finirons sur un test avec des captures d'écran à l'appuis.

Malgré le fait que le C/C++ ne soient pas des langages orientés objet (POO) mais très proches de la machine, on peut si on organise de manière rigoureuse le code atteindre quelque chose qui s'y approche, c'est ce que j'ai essayé de faire afin d'obtenir une vision clair du projet dans son ensemble. Ce paradigme permet de représenter de manière conceptuel son projet en créant principalement des structures et des classes, c'est à dire l'instanciation d'objet possédant des attributs.

A. Fichier principal

Le fichier principal de l'Arduino, c'est à dire le fichier qui exécute toutes les autres fonctions est le fichier `PPL.ino`, il est absolument essentiel. On y retrouve de manière caractéristique les méthodes :

- `setup()` qui permet de déclarer les sorties numériques (le buzzer), commencer la communication par le port série avec le capteur d'empreintes, paramétrer la transmission à 9600 bits.s^{-1} (vitesse standard de transmission), scanner est ici une instance de la classe du capteur d'empreintes connecté sur les ports 10 en réception (RX) et 11 en transmission (TX) selon le schéma de câblages :

```
void setup()
{
  Serial.begin(DEFAULT_TRANSMISSION_SPEED);
  print_debug("PPL onboard software - v.0.5 (instable)");

  assignPins();
  scanner.Open();
}
```

Fig. 10 - Fonction setup()

- `loop()` qui exécute à la façon d'une boucle while tout ce qui s'y trouve. Ici elle vérifie le statut de sécurité et en particulier si le capteur d'empreintes a été déverrouillé via des conditions booléennes : si le statut est verrouillé & si le capteur d'empreintes subit une pression & si le doigt reconnu appartient à l'ensemble connus alors on peut déverrouiller

et changer le statut de sécurité. Par ailleurs, elle évalue le statut du port série, si des données sont entrantes (`Serial.available() > 0`) alors elle les traite et appelle l'interpréter de commandes en transmettant les données reçues en paramètres sous forme d'un tableau unidimensionnel de caractères facilitant ainsi le traitement de chacun d'eux.

J'ai utilisé dans ce fichier de nombreuses fonctions auxiliaires renvoyant des données booléennes en majorité afin d'accentuer la lisibilité du code par quiconque.

B. L'interprétation des commandes

L'interprétation des commandes est un processus complexe et cruciale qui doit s'appuyer sur le protocole défini plus haut. Toute cette sous-section va être axé sur l'étude approfondie du fichier CommandsInterpreter.cpp. *D'ailleurs, il est important d'expliciter ce paradigme des fichiers à extensions 'h' et 'cpp'. Les fichiers 'h' sont des fichiers de déclarations qu'on nomme les headers, ils servent à définir les fonctions, variables et structures qui doivent être présents, on y déclare simplement le type : s'il s'agit d'une méthode, d'une fonction, son accessibilité : public, private (accessible seulement à l'intérieur du fichier); ils agissent comme maitre d'oeuvre lors de la compilation du programme en créant des liens entre les fichiers (les linkers). Les fichiers 'cpp' représentent quant à eux toute la partie fonctionnelle en accord avec le contrat du header.* (ANNEXE ?)

La méthode majeure ici est bien entendu `str_command_router` :

```
void CommandsInterpreter::str_command_router(char data[], unsigned int data_size) {
```

Fig. 11 - Concept de la méthode str_command_router : CommandsInterpreter.h

Elle est appelée depuis la méthode loop qui reçoit en temps réel les commandes depuis le software manager. Elle reçoit en paramètres les caractères individuelles sous forme d'un tableau unidimensionnel, puis la taille de se tableau pour éviter les problèmes de pointeurs. Elle va inspecter caractère par caractère le tableau `data[]` pour exprimer la forme conçue en **p.11**. On y définis de nombreuses variables :

```
String command = "";
unsigned int parameters_count = parametersCount(data, data_size);
String parameters[parameters_count];
unsigned short spaces_iterator = 0;
bool submiting_quote_in_progress = false;
bool submiting_space_in_param = false;
```

Fig. 12 - Méthode str_command_router : CommandInterpreter.cpp

La fonction `parametersCount` est une fonction auxiliaire de traitement de caractères qui permet de récupérer le nombre de paramètres (sous la forme d'un entier non signé : $0 \leq x \leq \sim 2e9$) en suivant la règles des espaces définie dans le concept **p.11**. Elle se base sur la

recherche des indices de positions des « » et d'une suppression des caractères dans cet intervalle en incrémentant le nombre de paramètres, ainsi suit une méthode plus classique d'analyse où les espaces sous forme décimales sont comptés. Nous sommes obligés de procéder au préalable de la manière suivante, car on ne peut pas initialiser un tableau d'une longueur inconnue en C/C++, ici `parameters[]`. S'en suit une analyse syntaxique complexe, dans une boucle `for` dans l'intervalle `[0; (data_size - 1)]` pour éviter le caractère d'échappement du type char : `'\0'`. Le caractère à l'indice *i* du tableau est récupéré puis on détermine via des variables booléennes si on est dans un cas particulier pour le paramètre p_k, c'est à dire s'il s'agit d'un paramètre sous la forme « hello world », on construit ainsi le tableau des paramètres. Après détermination de la commande et des paramètres, à partir d'une structure conditionnelles on appelle la fonction correspondante à la commande en vérifiant la longueur des paramètres, tel que comme sur le **Fig. 13**.

```
else if (command == INCOMING_PAYLOAD_ADD)
{
  if (parameters_count != 3)
  {
    return;
  }

  String website = parameters[0];
  String username = parameters[1];
  String password = parameters[2];

  outgoing_payload_add(website, username, password);
}
```

Fig. 13 - Structure conditionnelle méthode str_command_router : CommandsInterpreter.cpp

Toutes les méthodes / fonctions appelées correspondantes à une commande sont majoritairement des appels vers le module de gestion de données et qui renvoient le résultat sous la forme d'une commande « outgoing » via la méthode `Serial.println(str)` vers le software manager selon l'énumération des commandes faites dans le fichier CommandsEnum. Nous allons donc désormais étudier la gestion de données, c'est-à-dire la sauvegarde, l'édition et la suppression au sein de l'EEPROM en respectant le concept de Payload.

C. Gestion des données

La gestion des données est prise en charge par le concept de `Payload` qui définit trois attributs, le website, l'identifiant et le mot de passe. Conceptuellement, cette classe se définie comme décrit dans le **Fig. 14**, on y trouve deux attributs en plus : `head_start_address` et `head_end_address` (entiers non signés, car toutes les adresses > 0), il s'agit de l'intervalle de définition du Payload dans la mémoire, sa zone allouée. On y trouve le constructeur de la classe, c'est à dire la méthode qui va être exécutée lors de l'instanciation d'un objet `Payload`. Nous allons approfondir notre étude dans la description du constructeur et de la fonction `standard_malloc()`, les autres fonctions seront intégralement commentées dans les sources.

```
class Payload
{
  public:
    String website;
    String username;
    String password;

    unsigned int head_start_address;
    unsigned int head_end_address;

    Payload(unsigned int address, bool end_after_completing_website);

    bool standard_malloc();
    bool mralloc(String website, String username, String password);
    bool deletion_malloc();

    unsigned int payload_size();

    void uncipher();
    void cipher();
};
```

Fig. 13 - Payload.h

Le constructeur Payload prend deux paramètres (**Fig. X**) l'adresse de départ du payload, son `head_start_address`, puis une variable booléenne qui sert à continuer la recherche après avoir trouvé les valeurs hypothétiques du `website`, c'est utile pour récupérer l'existence d'un payload sans avoir à charger en mémoire toutes les données :

```
Payload(unsigned int address, bool end_after_completing_website);
```

Fig. 14 - Constructeur classe Payload : Payload.h

Ce constructeur a pour rôle d'aller chercher en mémoire les valeurs dans l'intervalle de définition et de les assigner aux attributs publics website, username, password. Il s'appuie sur une boucle qui explore l'EEPROM depuis l'adresse k_i = (head_start_address + 1), avec la librairie officielle et la fonction `EEPROM.read(`k_i`)`.

```
for (unsigned int i = iterator_start; i < eeprom_length; i++)
```

On peut instantanément arrêter la boucle si à k_{i-1} la valeur n'est pas un pattern de reconnaissance et en particulier un `head_start_pattern`. Ainsi, selon un processus itératif, on détermine les bornes de nos données, à chaque fois qu'un `head_data_start_pattern` est aperçu on incrémente un itérateurs de head_data_start_pattern ainsi, lorsqu'on est à i = 0, c'est l'attribut website, i = 1 : username, i = 2 : password (en combinant avec un intégrateur head_data_stop_pattern on peut ainsi borner les données). En plaçant simplement les valeurs dans un char on obtient une représentation du caractère de la via la forme décimale par conversion automatique

avec le tableau ASCII, il suffit dès lors de concaténer chaque caractère à l'attribut correspondant.

On va désormais s'intéresser à la fonction d'allocation dans la mémoire : `standard_malloc()`. Cette fonction alloue dans la mémoire sans se préoccuper de ce qui est contenu dans à kn, elle doit donc être utilisée avec précaution pour éviter tout risque de corruption de la mémoire. On a défini arbitrairement que chacun des attributs ne devait pas dépasser une taille de 64 caractères afin d'avoir des longueurs constantes pour l'initialisation de tableaux de traitements… Si une des variables a une taille > à 64, la fonction retourne un booléen false et s'arrête donc.

```
char payload_chars[3][64];
```

Fig. 15 - Tableau bi-dimensionnel de caractères

On crée ensuite un tableau bi-dimensionnelle avec 3 colonnes de 64 cases chacune : 3*64, pour traiter individuellement chaque caractère. Chacune des colonnes représente un attribut en particulier.

```
website.toCharArray(payload_chars[0], (payload_lengths[0] + 1));
username.toCharArray(payload_chars[1], (payload_lengths[1] + 1));
password.toCharArray(payload_chars[2], (payload_lengths[2] + 1));
```

Fig. 16 - Conversion string -> char[] : standard_malloc() => Payload.cpp

On convertie ainsi chaque string en tableau de char qu'on vient placer dans le tableau bi-dimensionnel aux indices : 0, 1, 2 avec une `taille += 1` pour le caractère d'échappement non pris en compte par les fonctions de conversions ici.

```
unsigned int address = head_start_address;
print_debug("Writing head_start_pattern at : 0x" + String(address));
EEPROM.write(address, PASSWORD_BLOCK_HEAD_START_PATTERN); //block start
address += 1;
```

Fig. 17 - Écriture pattern head_start : standard_malloc() => Payload.cpp

Puis on commence par écrire à head_start_address le pattern de reconnaissance head_start_pattern. S'en suit une double boucle `for` factorisé qui permet de naviguer dans les trois indices du tableau bi-dimensionnel de manière itérative avec les indices i et j deux entiers non signés. On a juste à vérifier quand j = 64 ou dans le cas contraire quand j = 0, dès lors on peut écrire respectivement les head_data_start_pattern et head_data_stop_pattern, sinon le caractère convertir en format décimal via le cast en nombre entier : (int).

```
for (unsigned int i = 0; i < 3; i++) //Factorised loop to write payload attributes into the EEPROM
{
    print_debug("FW i = " + String(i));
    for (unsigned int j = 0; j <= payload_lengths[i]; j++)
    {
        print_debug("FW j = " + String(j));
        unsigned short ascii_index = (int)payload_chars[i][j]; //get the decimal representation of the char located at [i][j]
        print_debug("ascii_index = " + String(ascii_index) + " | payload_chars[" + String(i) + "][" + String(j) + "]");

        if (j == payload_lengths[i])
        {
          print_debug("Writing block_data_end_pattern at 0x" + String(address));

          EEPROM.write(address, PASSWORD_BLOCK_DATA_END_PATTERN);
        }
        else
        {
          if (j == 0)
          {
            print_debug("Writing block_data_start_pattern at 0x" + String(address));
            EEPROM.write(address, PASSWORD_BLOCK_DATA_START_PATTERN);
            address += 1;
          }

          print_debug("Writing ascii index : 0x" + String(ascii_index) + " at 0x" + String(address));
          EEPROM.write(address, ascii_index);
        }
```

Fig. 18 - Boucle d'écriture factorisé : standard_malloc()=>Payload.cpp

Je pense désormais avoir suffisamment explicité mon travail sur cette partie-ci du projet, je ne peux malheureusement pas parler de toutes les fonctions. Je pense avoir pu montrer les fonctions à enjeux essentiel quand au respect du cahier des charges. Il est important de rappeler que tout le code est commenté et que je me tiens disponible à répondre à n'importe quelle question concernant mon code. Je vais désormais parler de ma contribution au software manager.

C. Contribution / aide software manager

Afin d'aider Donatien face à des concepts la majorité du temps non vus en cours, j'ai dû l'aider dans le but d'améliorer objectivement le projet. J'ai donc conçu l'interface de communication conceptualisée par la classe SerialCom qui sert à envoyer les données vers le port série et les recevoir grâce à la librairie pyserial.

Au sein de son constructeur, on établie les paramètres de transfert identiques à l'Arduino pour éviter les conflits. Si le port est déjà ouvert on force l'accès et on le ferme pour le réouvrir. On définit la variable serial avec l'instance de la classe Serial (du module pyserial) afin de pouvoir l'utiliser ailleurs de manière publique.

Fig. 19 - Constructeur classe SerialCom : SerialCom.py

```
def __init__(self):
    default_transmission_rate = 9600  # baud.s^-1
    default_chip_location = '/dev/cu.usbmodem1421'

    ser = serial.Serial()
    ser.baudrate = default_transmission_rate
    ser.port = default_chip_location
    ser.timeout = 10
    if ser.isOpen():
        ser.close()
    ser.open()

    self.serial = ser
```

La seule difficulté ici a été de gérer les String, car avec Python 3 il faut les encoder en binaire pour les envoyer via le port série, **Fig. 20**. La fonction wait_read est la fonction exécutée lors du second thread de lecture des commandes. Chaque String doit être nettoyé des caractères de fins et de nouvelles lignes envoyés par le port série, **Fig. 21**.

```
def send(self, raw_command):
    if type(raw_command) is not str:
        raise Exception('Bad data handled, must be str')

    self.serial_not_init_check()
    self.serial.write(str.encode(raw_command))
```

Fig. 20 - Fonction send : SerialCom.py

```
def wait_read(self):
    while True:
        try:
            self.serial_not_init_check()

            incoming_data = self.serial.readline().decode()
            incoming_data = incoming_data.replace('\n', '')
            incoming_data = incoming_data.replace('\r', '')

            command_router(incoming_data)
        except KeyboardInterrupt:
            print("wait_read while stopped")
            break
```

Fig. 21 - Fonction wait_read : SerialCom.py

J'ai donc crée le système de multi-threading qui exécute la fonction « wait_read » qui contenant une boucle s'exécute jusqu'à temps que le programme s'arrête. Le thread est protégé d'une structure en try/catch permettant d'anticiper la levée d'erreur(s) sans empêcher de faire fonctionner le programme principal, **Fig. 22**.

```
def serial_read():
    try:
        # global serialcom_instance
        Vars.serialcom_instance = SerialCom()
        t1 = threading.Thread(target=Vars.serialcom_instance.wait_read, args=())
        t1.start()
    except Exception as e:
        print(str(e))
```

Fig. 22 - Fonction serial_read() : multi threading : __init__.py

J'ai par ailleurs pu développer le système d'interprétation de commandes qui est globalement le même qu'en C avec un algorithme plus simple répondant toujours aux concepts du protocole définit dans le cahier des charges.

D. Tests

Le statut fonctionnel de la sauvegarde des données peut être confirmé par un affichage des valeurs en hexadécimale (base 16) dans les bornes d'un payload via le port série. Ainsi, en plaçant les bits dans un éditeur hexadécimale on obtient le **Fig. 23**.

```
01027477 69747465 722E636F 6D03026A 73616E67 6C693372 03027563 616E7473 65656D79   twitter.com  jsangli3r  ucantseemy
70617373 776F7264 0304                                                            password
```

Fig. 23 - Échantillon de valeurs hexadécimales, représentant des données tests

Pour des données entrantes définies par la création d'une instance de l'objet Payload à la prochaine adresse disponible dans la mémoire telle que sur le **Fig. 24**.

```
Payload p(nextAvailableAddress(), true);
p.website = "twitter.com";
p.username = "jsangli3r";
p.password = "ucantseemypassword";

p.standard_malloc();
printBornedValues(0, 100);
```

Fig. 24 - Création des données tests

Nous allons désormais essayer de récupérer ces données via le software manager, **Fig. 25.**

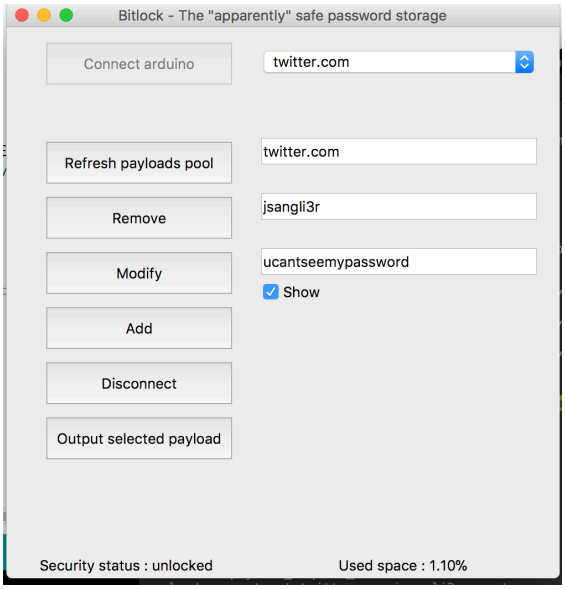


Fig. 25 - Interface du software manager

Les données étant équivalentes on peut certifier le système comme fonctionnel. Chacune des fonctions n'est pas testé ici mais elles sont en mesure de fonctionner sans problème, nous les exposerons lors des présentations orales.

Bilan / Perspectives/ Diffusion

En conclusion, faire ce projet fut une expérience très enrichissante pour nous deux. Nous avons appris énormément de choses, de concepts, de principes et avons pu expérimenter tout au long de la réaliser tout en collaborant.

Si nous devions améliorer le projet, nous finirions toute la partie sur la cryptographie qui est essentielle au fonctionnement nominale et qui normalement fonctionne sur la papier ! On pourrait par ailleurs, créer une boite, un support imprimé en 3D (dans un Fablab par exemple) à l'ensemble du système onboard pour maintenir le capteur d'empreintes après miniaturisation avec remplacement de l'Arduino par un contrôleur beaucoup moins gros, moins cher, avec seulement le strict minimum pour gérer une EEPROM. On pourrait sans aucun doute donner plus de soin à l'interface graphique du logiciel, voire faire une extension dans les navigateurs écrite en javascript.

Quite à diffuser le projet, pourquoi ne pas se lancer dans une campagne Kickstarter dès que le temps le permettra afin de faire partager ce projet au plus grand nombre !

Merci beaucoup à vous d'avoir pris le temps de lire ce rapport qui je le conçois est assez long.

Me contacter :

Annexe

court terme[1] = certes, il est assez peu positif de regarder un problème sur le court terme c'est pourquoi à l'avenir nous mettrons à profit nos connaissances dans le but d'apporter une solution durable et évolutive.

PyQt[2] = https://sourceforge.net/projects/pyqt/

pyserial[3] = https://github.com/pyserial/

pyautogui[4] = https://github.com/asweigart/pyautogui

pyperclip[5] = https://github.com/asweigart/pyperclip

PyCharm[6] = https://www.jetbrains.com/pycharm/

Arduino IDE[7] = https://www.arduino.cc/en/Main/Software

Trello[8] = https://trello.com/

Bitbucket[9] = https://bitbucket.org/

Git[10] = https://git-scm.com/

Slack[11] = https://slack.com/intl/fr-fr

Projet équivalent[12] = https://hackaday.io/project/3555-zamek-the-offline-pocket-password-manager

BCrypt[13] = https://fr.wikipedia.org/wiki/Bcrypt