Projet Authenticator

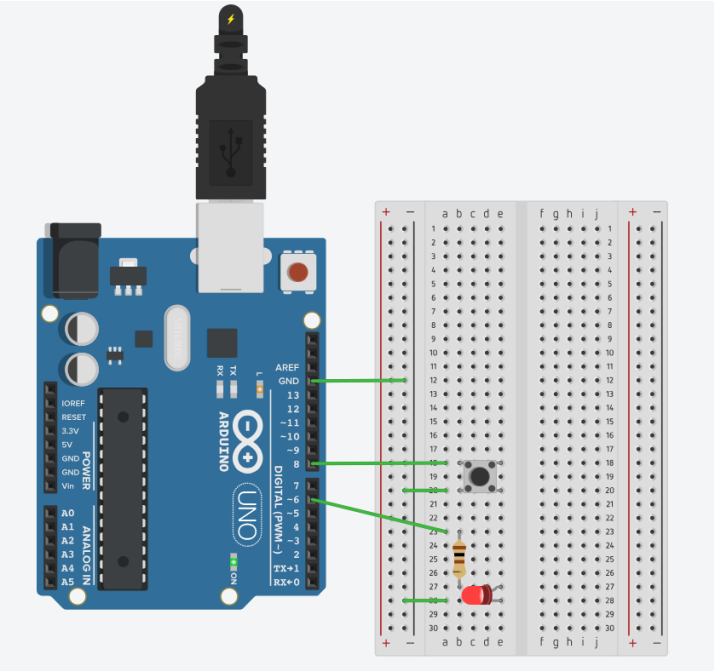
Equipe : `S4KOZ17`

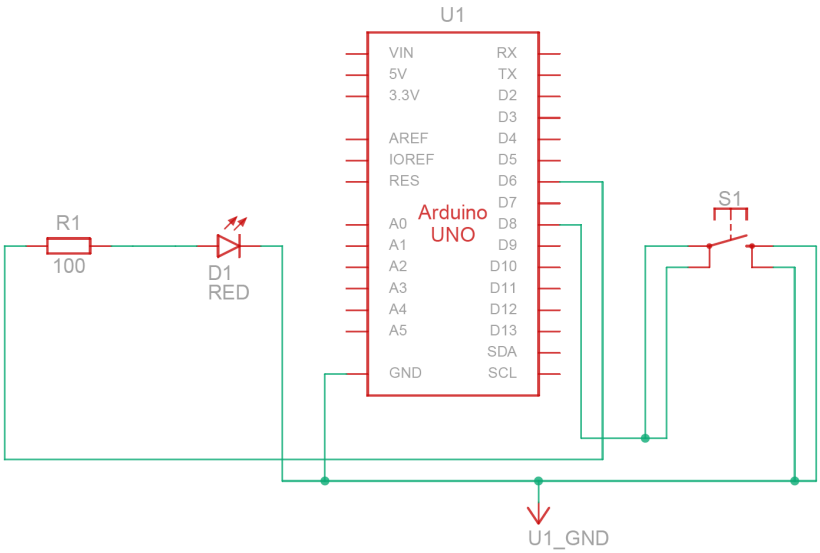
---

1. Schéma électrique

Sur l'Arduino, nous avons connecté une LED au pin `PD6 (digital pin 6)`, et un bouton-poussoir au pin `PB0 (digital pin 8)`.

Ci-dessous, vous trouverez une représentation du circuit ainsi que le schéma électrique correspondant :





2. Compilation

```
cd ./code/
# Compilation des programmes pour le microcontrôleur :
make # Compile le programme. Nécessite une confirmation manuelle pour l'authentification.
make test # Compile le programme pour le mode test. La confirmation pour l'authentification est automatique.
# Ces étapes sont intégrées dans le processus du makefile :
ls /dev/ | grep ttyACM0 # Étape optionnelle : vérifie si le microcontrôleur est correctement connecté et reconnu.
make clean # Étape optionnelle : nettoie
```

3. Résultats des tests

3.1. Dialogue avec Relying Party

> Le code d'erreur `6` signifie que la dernière commande n'a pas reçu le consentement de l'utilisateur.

```
INFO:root:Connect to device /dev/ttyACM0
Welcome to the Yubino shell. Type help or ? to list commands.

yubino > index
Salut utilisateur anonyme!
Créé un compte puis authentifie toi!
yubino > register s4KOZ17
INFO:root:Sending MAKE_CREDENTIAL command with hashed_app_id=23d93807be35983be22b0800014328b
cf7ada862
done
yubino > login s4KOZ17
INFO:root:Sending GET_ASSERTION command with hashed_app_id=23d93807be35983be22b0800014328bcf
7ada862 and challenge=d387a62a9c90981f73a965fdd4984866bc74fdada522da33a074be867358676fff08ce
42eb78cccaaf4a7e3a5a617f49ceb0fe487e3da9a0f4bb2e2079fcec64
done
yubino > index
Bien ouej s4KOZ17, tu as réussi à t'authentifier :-)
yubino > logout
done
yubino > index
Salut utilisateur anonyme!
Créé un compte puis authentifie toi!
yubino > login s4KOZ17
INFO:root:Sending GET_ASSERTION command with hashed_app_id=23d93807be35983be22b0800014328bcf
7ada862 and challenge=12e822061da88164ff261fb9f368d4cbc36af411728202ebc5c28e113303ee70ec273b
d2a582612a696370e0ee8a1c975b43b8a954f1d737a423ee9c7fc176e4
ERROR:root:Something bad happenned: error code 6
ERROR:root:Failed to get assertion: Device returned error code 6
failed
yubino >
```

Figure 1: Dialogue avec Relying Party

3.2. Résultats de : `python3 setup.py test`

```
(.env) cg@cg-HP:~/Downloads/PROJET/yubino-client$ python3 setup.py test
running test
WARNING: Testing via this command is deprecated and will be removed in a future version. Users looking
 for a generic test entry point independent of test runner are encouraged to use tox.
running egg_info
writing yubino.egg-info/PKG-INFO
writing dependency_links to yubino.egg-info/dependency_links.txt
writing entry points to yubino.egg-info/entry_points.txt
writing requirements to yubino.egg-info/requires.txt
writing top-level names to yubino.egg-info/top_level.txt
reading manifest file 'yubino.egg-info/SOURCES.txt'
writing manifest file 'yubino.egg-info/SOURCES.txt'
running build_ext
test_bad_command (tests.device.TestDevice) ... ok
test_get_assertion (tests.device.TestDevice) ... ok
test_get_assertion_app_unknown (tests.device.TestDevice) ... ok
test_get_assertion_multiple_creds (tests.device.TestDevice) ... ok
test_make_credentials (tests.device.TestDevice) ... ok
test_make_credentials_already_existing (tests.device.TestDevice) ... ok
test_make_credentials_full (tests.device.TestDevice) ... ok
test_reset (tests.device.TestDevice) ... ok

----------------------------------------------------------------------
Ran 8 tests in 44.703s

OK
```

Figure 2: Tests fournis

4. Stockage des données (EEPROM)

Pour rappel, la mémoire EEPROM a une capacité de 1024 octets. Une application occupe 57 octets, répartis entre le `credential_id` (16 octets), `app_id` (20 octets) et `private_key` (21 octets). Ainsi, on peut stocker au maximum 17 applications.

À l'adresse 0, nous enregistrons le nombre d'applications `nbApps`. Par conséquent, pour ajouter une application, nous la stockons à l'adresse $(nbApps \times 57) + 1$, en suivant l'ordre `credential_id`, `app_id`, `private_key`.

5. Fonction RNG : `get_alea_from_Timer0`

Initialement, nous utilisions l'ADC (Analog-to-Digital Converter) pour générer des nombres aléatoires. Cependant, cette méthode s'est avérée chronophage et imposait une action supplémentaire à l'utilisateur. Pour remédier à cela, nous avons opté pour l'utilisation du `timer0` à la place.

6. Configuration du BAUD rate 115200

Au début, l'utilisation de la formule de calcul de l'`UBRR` (USART Baud Rate Register) entraînait des erreurs d'arrondis. Pour résoudre ce problème, nous avons consulté la documentation d'`avr-libc` (en particulier `setbaud.h`) afin d'adapter le code de l'UART.

7. Fonctionnalités

- **Demande de `consentement`** : La LED clignotante connectée au pin `PD6` indique à l'utilisateur qu'un consentement est nécessaire. L'utilisateur doit appuyer sur le bouton connecté au pin `PB0` pour donner son consentement. Cette action est gérée par la fonction `get_consent`.
- **Fonction `make_credentials`** : Cette fonction prend en entrée `app_id` et vérifie s'il existe déjà dans la mémoire non volatile.
  - **Si existant** : La fonction lit le `credential_id` associé et le retourne, avec une clé publique contenant des données aléatoires (données résiduelles du tableau).
  - **Si non existant** : La fonction demande d'abord le consentement via `get_consent`. Ensuite, elle génère les clés à l'aide de la bibliothèque `micro-ecc` et produit un `credential_id` en utilisant notre fonction RNG. Elle stocke ensuite le `credential_id`, `app_id`, la clé privée et `nbApps` incrémenté dans la mémoire EEPROM. Par la suite, le `credential_id` et la clé publique sont transmis au client Yubino.
- **Fonction `get_assertion`** : En plus de l'`app_id`, cette fonction prend un `client_data_hash` à signer. Elle commence par vérifier si l'application est présente dans l'EEPROM en parcourant les `app_id`.
  - **Si existante** : La fonction demande le consentement, puis signe avec la clé privée récupérée de l'EEPROM. La signature et le `credential_id` sont ensuite envoyés au client Yubino.
  - **Si inexistante** : La fonction retourne l'erreur `STATUS_ERR_NOT_FOUND`.
- **Commande `COMMAND_LIST_CREDENTIALS`** : Lors de la réception de cette commande, il n'est pas nécessaire de demander un consentement. La fonction récupère le nombre d'applications existantes dans l'EEPROM et retourne progressivement les `credential_id` et `app_id` associés. Cette commande ne rencontre jamais d'échec.
- **Réception de la commande `COMMAND_RESET`** : Un consentement est demandé via `get_consent`. Si l'utilisateur consent, toutes les données de la mémoire non volatile sont remplacées par des zéros.

8. Difficultés rencontrées

Nous avons rencontré un blocage de 5 jours, causé par des comportements incohérents dus à une configuration matérielle avant l'établissement de la communication UART. Le problème principal était la configuration du registre `PRR` (Power Reduction Register) à `0xFF`. Cette valeur a pour conséquence de désactiver tous les périphériques intégrés, y compris le périphérique `PRUSART0` essentiel pour la communication UART.