commande de porte de garage par Bluetooth à réponse rapide

ouvrez votre porte avec votre smartphone

Stephan Lück (Allemagne)

Il m'arrive souvent d'avoir besoin d'ouvrir la porte de mon garage alors que je n'ai pas l'émetteur de télécommande 433 MHz en poche. J'ai donc construit un petit récepteur Bluetooth Low Energy (BLE) qui peut être commandé depuis une application Android sur mon smartphone.

L'une des particularités de ce projet est qu'il utilise des unités de données de protocole (PDU, *Protocol Data Units*) de publication BLE, ce qui évite les délais d'établissement de la connexion. Le système est donc aussi rapide qu'une paire émetteur/récepteur classique. La communication entre le smartphone et le récepteur est unidirectionnelle et sécurisée par un code d'authentification de message obtenu par hachage (HMAC, *hash-based message authentication code*) [1]. L'idée était de faire simple, autant pour le matériel que pour le logiciel.
Je publie ce projet dans l'intention qu'il soit utilisable tel quel ou comme base de projets similaires. Le code source est disponible sur GitHub [2].

Le protocole

Pour transmettre une commande, on utilise les PDU de publication BLE, qui sont décrites dans le volume 6, partie B, section 2.3 de la spécification Bluetooth [3]. Pour des raisons de compatibilité, on utilise les PDU courtes, à l'ancien format, qui contiennent un champ *AdvData* de 31 octets au maximum. Le champ *AdvData* contient une séquence d'éléments de données de publication (AD, *Advertising Data*) [4] (cf. volume 3, partie C, section 11 de la spécification Bluetooth). Un élément AD possible est l'élément de données de service (cf. supplément de spécification du noyau Bluetooth, partie A, section 1). Cet AD contient un UUID (*Universal Unique Identifier*) de service suivi d'un nombre arbitraire d'octets.
Cette structure AD est utilisée pour transmettre une commande à distance. L'UUID de 128 bits correspond à un identifieur (ID) d'émetteur unique. En d'autres termes, chaque émetteur définit un service BLE spécifique. Les données jointes à l'UUID sont structurées comme suit, et sont représentées simplifiées sur la **figure 1** :

```
typedef struct {
uint8_t cmd; /* commande (actuelle-
   ment toujours 0) */
uint8_t seq_no[3]; /* numéro de
   séquence (big endian) */
uint8_t digest[4]; /* quatre premiers
   octets de HMAC-SHA256 */
} gd_message_t;
```

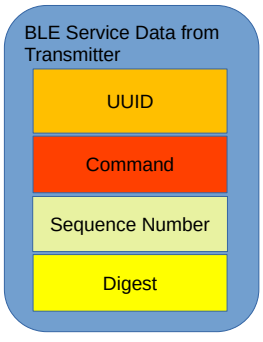


Figure 1. Structure de données simplifiée.

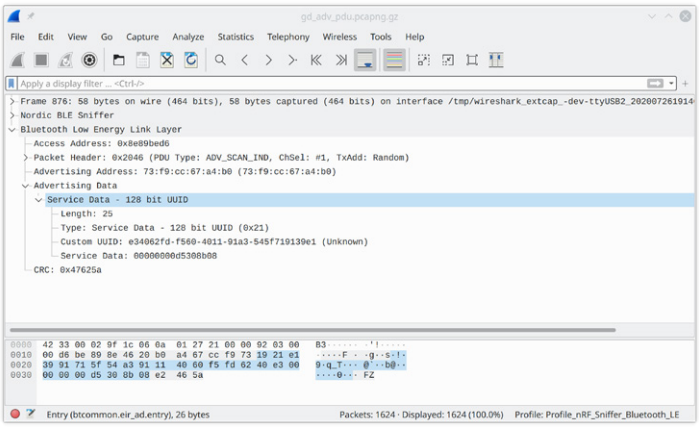


Figure 2. PDU capturé par Wireshark.

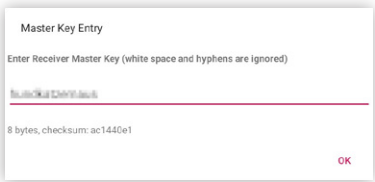


Figure 3. Démarrage de l'application et génération de la clé.

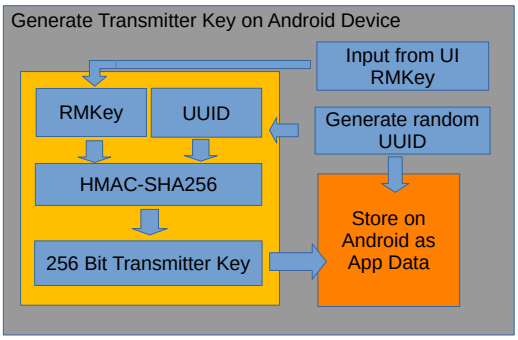


Figure 4. Flux pour le calcul de la clé de l'émetteur.

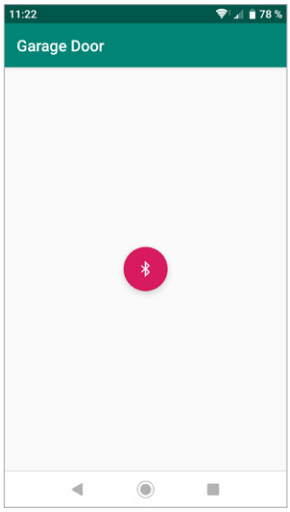


Figure 5. L'application est prête à être utilisée.

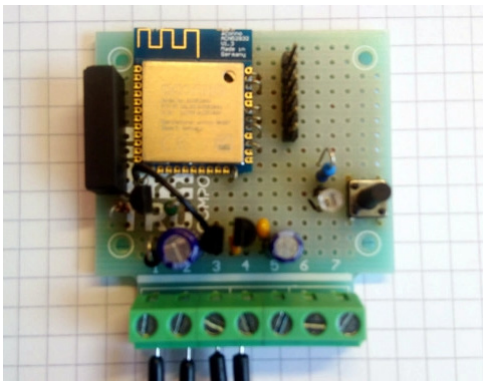


Figure 6. Prototype prêt.

Le champ *digest* permet d'authentifier le message. Le numéro de séquence permet de détecter les PDU dupliquées et d'éviter les attaques par répétition. Les PDU de publication BLE n'étant pas cryptées, il est possible que d'autres utilisateurs reçoivent à la fois l'UUID et les données supplémentaires. Le *digest* est calculé à partir d'une clé d'émetteur, laquelle dérive d'une clé principale de récepteur (RMkey) stockée dans le récepteur, basée sur l'UUID de l'émetteur.

```
transmitter_key = HMAC-SHA256(RMkey,
transmitter_UUID)
```

Un exemple de PDU de publication est illustré dans la capture d'écran Wireshark de la **figure 2.**

L'émetteur

L'émetteur est une simple application Android nécessitant le niveau 21 de l'API (correspondant à Android 5 « Lollipop »). Au premier démarrage, l'application calcule un UUID aléatoire et affiche une boîte de dialogue de configuration pour saisir la RMkey sous la forme d'une valeur codée en base32 (**figure 3**). Le dialogue de configuration montre la longueur de la clé et une somme de contrôle CRC32 pour aider l'utilisateur à éviter les fautes de frappe lors de la saisie de la clé. L'application calcule la clé de l'émetteur à partir de l'UUID et de la RMkey qui est jetée. Le tuple (UUID, clé de l'émetteur), correspondant à l'identité de l'émetteur, est stocké de manière permanente. Cette identité est perdue lorsque l'application est désinstallée ou que les données de l'application sont supprimées manuellement. Le principe du calcul est illustré à la **figure 4.**

Une fois configurée, l'application affiche un bouton rouge (**figure 5**). En cliquant sur le bouton, une procédure de publication BLE est configurée pour répéter la transmission de l'AD de données de service décrite ci-dessus pendant une durée de quelques secondes.

Le récepteur

Le récepteur a été construit sur une carte de prototypage autour du module BLE aconno ACN52832 [5] (**figure 6**). Il comporte un microcontrôleur Nordic Semiconductor NRF52382, un bouton poussoir, une LED et un relais à connecter à une motorisation de porte de garage. Le récepteur a besoin d'une alimentation de 9 V à 24 V DC. Si possible, la tension d'alimentation peut être prise sur la motorisation de la porte de garage. Sinon, une simple alimentation murale fera aussi bien l'affaire. Vous trouverez le schéma à la **figure 7**.

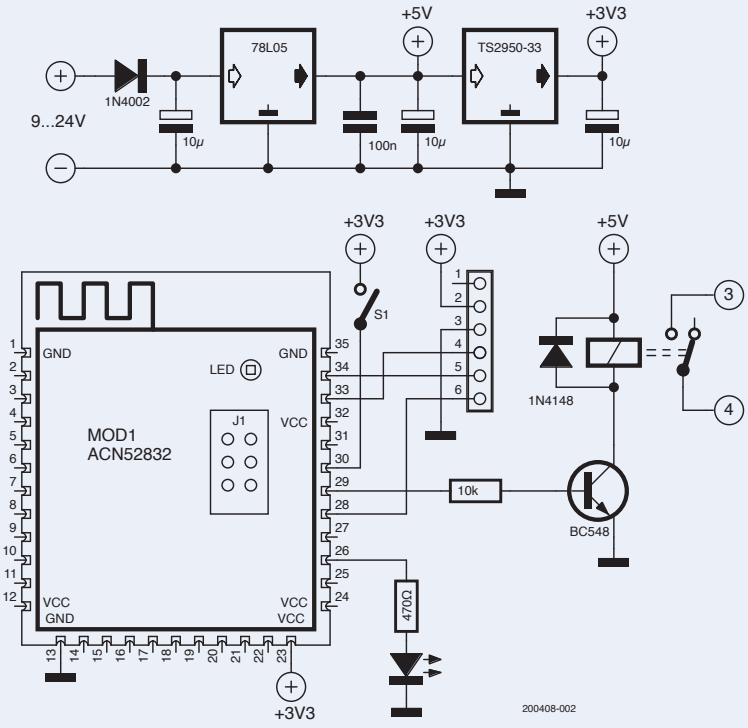


Figure 7. Schéma du récepteur.

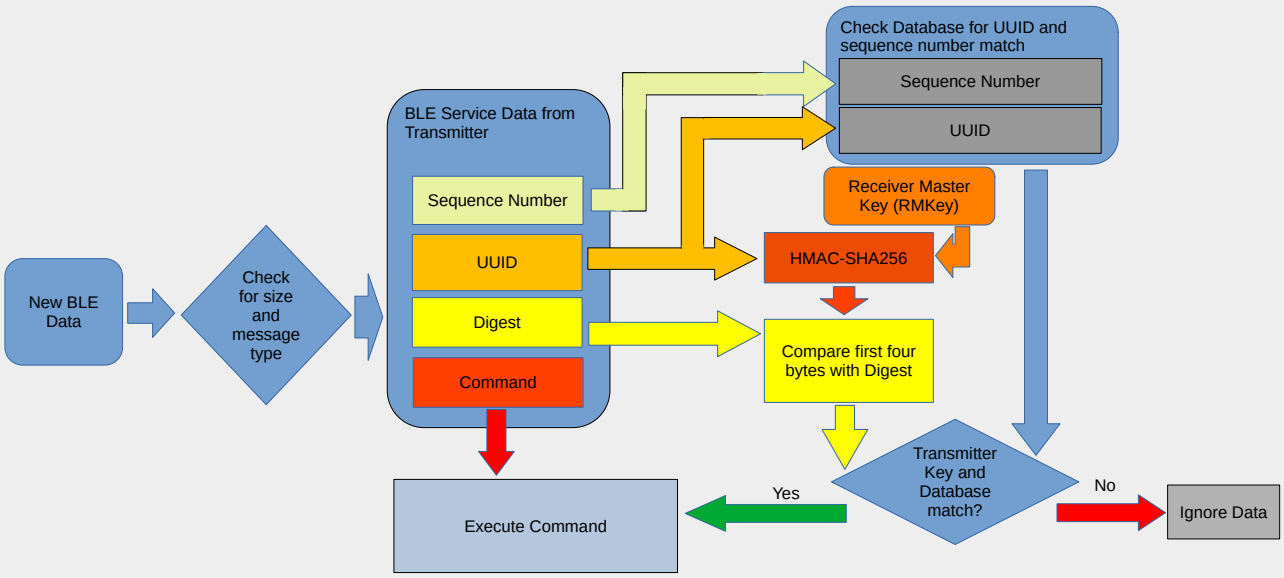


Figure 8. Flux de réception des données BLE.

Lors de la génération du logiciel pour le microcontrôleur, un script Python crée le fichier *rxm_key.bin* qui contient la clé RMkey de 20 octets. De plus, une représentation textuelle de la clé est créée et stockée dans le fichier *_build/rxm_key.txt*, qui contient les informations à saisir dans l'application de l'émetteur lors du premier démarrage.

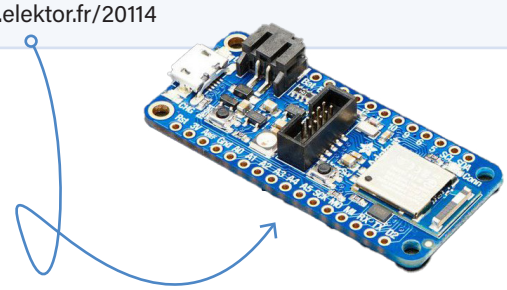
Le récepteur gère une base de données des UUID et des numéros de séquence des émetteurs. Pour ajouter un nouvel émetteur, il faut appuyer sur le bouton et ensuite activer l'émetteur à ajouter. Si l'on appuie sur le bouton pendant plus de cinq secondes, la base de données est effacée et le récepteur ne répond plus à aucun émetteur.

Lorsque le récepteur reçoit une PDU de publication contenant la structure AD de données de service décrite ci-dessus, il vérifie la présence de l'UUID de l'émetteur dans sa base de données et que le numéro de séquence est correct (**figure 8**). En cas de succès, le relais est activé pendant une seconde.

200408-04 – VF : Helmut Müller

PRODUITS

- **Adafruit CLUE – nRF52840 Express avec Bluetooth LE (SKU 19512)**
  www.elektor.fr/19512
- ***Livre en anglais « Android App Development for Electronics Designers », Dogan Ibrahim,* (SKU 18687)**
  www.elektor.fr/18687
- **Dongle USB nRF52840 MDK de makerdiary avec boîtier (SKU 19252)**
  www.elektor.fr/19252
- **Adafruit Feather nRF52840 Express (SKU 20114)**
  www.elektor.fr/20114



Des questions, des commentaires ?

Contactez l'auteur via GitHub [2] ou contactez Elektor (redaction@elektor.fr).

LIENS

[1] HMAC (Wikipedia) : https://fr.wikipedia.org/wiki/HMAC
[2] Dépôt GitHub : https://github.com/kiffie/ble-garage-door
[3] Bluetooth SIG : www.bluetooth.com/
[4] Principes de base des données de publication Bluetooth, Silicon Labs : https://bit.ly/silabs-bluetooth-ad
[5] ACN52832 par aconno : https://aconno.de/products/acn52832/