Systèmes embarqués pour objets connectés
SEOC2
Bluetooth sur ESP32

IUT de Béziers, dépt. R&T © 2018
http://www.borelly.net/
Christophe.BORELLY@umontpellier.fr

Bluetooth sur ESP32

- Bluetooth 4.2 **BR/EDR** and **BLE** dual mode controller
- Multi-connections in Classic BT and BLE
- Simultaneous advertising and scanning
- Class-1 (100mW), class-2 (2,5 mW) and class-3 (1mW) transmitter
- Enhanced Power Control
- +12 dBm transmitting power
- NZIF receiver with –97 dBm BLE sensitivity
- Adaptive Frequency Hopping (AFH)
- Standard HCI based on SDIO/SPI/UART
- High-speed UART HCI, up to 4 Mbps
- Synchronous Connection-Oriented/Extended (SCO/eSCO)
- CVSD and SBC for audio codec
- Bluetooth Piconet and Scatternet

Versions de bluetooth

- **BR** : Basic Rate (1 Mbit/s) v1.1 2002
- **EDR** : Enhanced Data Rate (2 and 3 Mbit/s) v2.0 2004
- **HS** : High Speed (jusqu'à 24 Mbit/s) v3.0 2009
- **LE** : Low Energy (1 Mbit/s ultra low power) v4.0 2010
- BT core specification v5.0 2016 (2822 pages)
  - BLE vol.6 page 2528 : 2 Mbit/s, portée plus grande

Bluetooth Low Energy

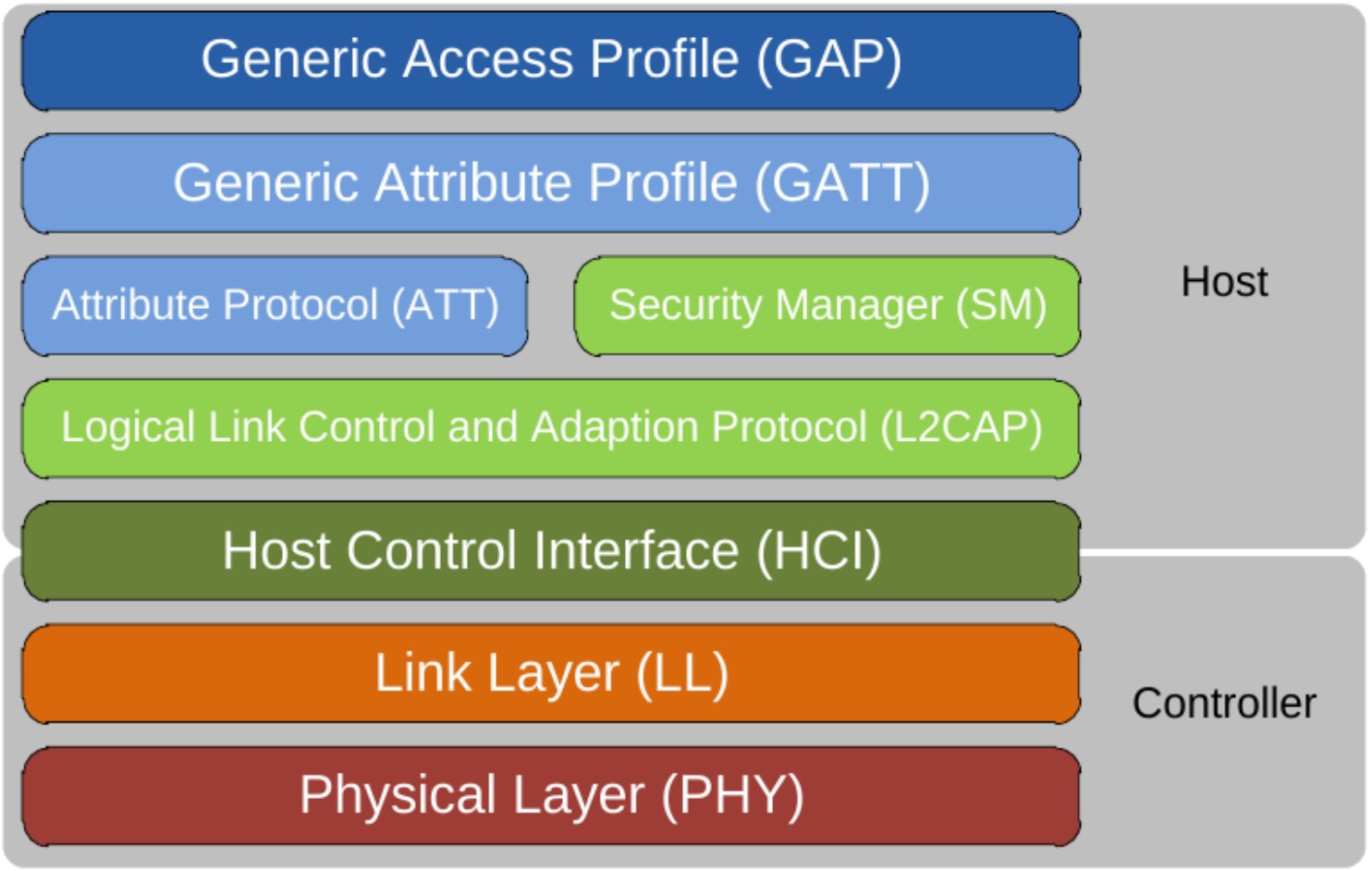
- Incompatible avec BT classique : BR/EDR
- Utilise la bande **ISM** (Industrial, Scientific and Medical) : 2402-2480 MHz
- 40 canaux de 2 MHz (3 canaux balises)
- Modulation GFSK (FHSS 1600 sauts/s)
  - Frequency Hopping Spread Spectrum
- Securité : 128 bits AES-CMAC

Support de BLE

- iOS5+
- Android 4.3+

- Apple OS X 10.6+
- GNU/Linux 3.4+ + BlueZ 4.93+
- Windows 8+ (XP, Vista and 7 => Bluetooth 2.1)
- ...

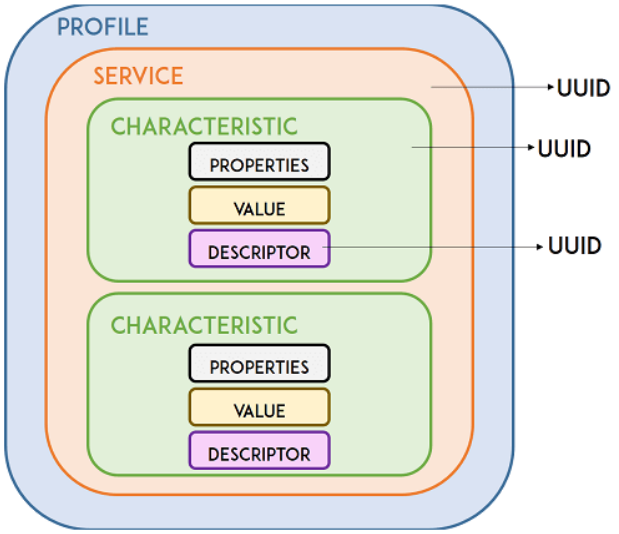
Pile Bluetooth



GAP

- **G**eneric **A**ccess **P**rofile
  - Découverte et gestion des connexions
  - Rôles :
    - **Broadcaster** (tx)
    - **Observer** (rx)
    - **Peripheral** (slave rx + tx)
    - **Central** (master rx + tx)
  - Paquets de 31 octets

GATT



- **G**eneric **ATT**ribute profile
  - Rôles : Client, Serveur ou les 2
  - Connexion BT requise et exclusive (bloque les « advertising » et les autres connexions)
  - Profile, Services, Characteristics, ...
  - Universally Unique IDentifier (UUID)
    - 16 bits (services BLE officiels) ou 128 bits
  - Propriétés :
    - Read, Write, WriteWithoutResponse
    - Notify, Indicate, Broadcast

Exemples de UUID officiels

- https://www.bluetooth.com/specifications/gatt/services
  - 0x1810 org.bluetooth.service.blood_pressure
    - 0x2A49 org.bluetooth.characteristic.blood_pressure_feature
    - 0x2A35 org.bluetooth.characteristic.blood_pressure_measurement
  - 0x180F org.bluetooth.service.battery_service
    - 0x2A19 org.bluetooth.characteristic.battery_level
    - 0x2A1B org.bluetooth.characteristic.battery_level_state
    - 0x2A1A org.bluetooth.characteristic.battery_power_state

ESP32 BLE support for Arduino

- https://github.com/nkolban/ESP32_BLE_Arduino
- BLE_server
- BLE_scan
- BLE_client
- BLE_write
- BLE_notify
- ...
- Pas d’exemples de sécurité pour l’instant...

BLE_server.ino (2 bugs)

```
#include <BLEDevice.h>
#include <BLEUtils.h>
#include <BLEServer.h>
#define SERVICE_UUID        "4fafc201-1fb5-459e-8fcc-c5c9c331914b"
#define CHARACTERISTIC_UUID "beb5483e-36e1-4688-b7f5-ea07361b26a8"
void setup() {
  Serial.begin(115200);
  Serial.println("Starting BLE work!");
  BLEDevice::init("ESP32"); // Maximum 5 chars !!!???
  BLEServer *pServer=BLEDevice::createServer();
  BLEService *pService=pServer->createService(SERVICE_UUID);
  BLECharacteristic *pCharacteristic=pService->createCharacteristic(
     CHARACTERISTIC_UUID,
     BLECharacteristic::PROPERTY_READ | BLECharacteristic::PROPERTY_WRITE );
  pCharacteristic->setValue("Hello World says Neil");
  pService->start();
  BLEAdvertising *pAdvertising=pServer->getAdvertising();
  pAdvertising->addServiceUUID(pService->getUUID());
  pAdvertising->start();
  Serial.println("Characteristic defined!");
}
void loop() { delay(2000); }
```

Client Bluetooth Linux

- Paquet Debian : bluez

```
# systemctl status bluetooth
...
# hciconfig
hci0:   Type: Primary  Bus: USB
        BD Address: 28:16:AD:B6:E4:EF  ACL MTU: 1021:4  SCO MTU: 96:6
        UP RUNNING PSCAN
        RX bytes:15285 acl:0 sco:0 events:2463 errors:0
        TX bytes:605920 acl:0 sco:0 commands:2459 errors:0
# hcitool lescan
LE Scan ...
24:0A:C4:1D:29:66 ESP32
24:0A:C4:1D:29:66 ESP32
```

Linux : gatttool

```
# gatttool -b 24:0A:C4:1D:29:66 -I
[24:0A:C4:1D:29:66][LE]> connect
Attempting to connect to 24:0A:C4:1D:29:66
Connection successful
[24:0A:C4:1D:29:66][LE]> characteristics
handle: 0x0002, char properties: 0x20, char value handle: 0x0003, uuid: 00002a05-0000-1000-8000-00805f9b34fb
handle: 0x0015, char properties: 0x02, char value handle: 0x0016, uuid: 00002a00-0000-1000-8000-00805f9b34fb
handle: 0x0017, char properties: 0x02, char value handle: 0x0018, uuid: 00002a01-0000-1000-8000-00805f9b34fb
handle: 0x0019, char properties: 0x02, char value handle: 0x001a, uuid: 00002aa6-0000-1000-8000-00805f9b34fb
handle: 0x0029, char properties: 0x0a, char value handle: 0x002a, uuid: beb5483e-36e1-4688-b7f5-ea07361b26a8
[24:0A:C4:1D:29:66][LE]> char-read-uuid beb5483e-36e1-4688-b7f5-ea07361b26a8
handle: 0x002a   value: 48 65 6c 6c 6f 20 57 6f 72 6c 64 20 73 61 79 73 20 4e 65 69 6c
[24:0A:C4:1D:29:66][LE]> char-read-hnd 0x002a
Characteristic value/descriptor: 48 65 6c 6c 6f 20 57 6f 72 6c 64 20 73 61 79 73 20 4e 65 69 6c
[24:0A:C4:1D:29:66][LE]> char-write-req 0x002a 73616c7574
Characteristic value was written successfully
[24:0A:C4:1D:29:66][LE]> char-read-hnd 0x002a
Characteristic value/descriptor: 73 61 6c 75 74
[24:0A:C4:1D:29:66][LE]> disconnect
[24:0A:C4:1D:29:66][LE]> exit
```

Décodage hexadécimal

```
echo -n 48 65 6c 6c 6f 20 57 6f 72 6c 64 20 73 61 79 73 20 4e 65 69 6c | tr -d ' ' | xxd -r -p
```

- Hello World says Neil

```
echo -n salut | xxd -p
```

- 73616c7574

Linux : bluetoothctl

```
$ bluetoothctl
Agent registered
[ESP32]# menu gatt
...
[ESP32]# list-attributes
...
Characteristic
        /org/bluez/hci0/dev_24_0A_C4_11_03_12/service0028/char0029
        beb5483e-36e1-4688-b7f5-ea07361b26a8
        Vendor specific
...
[ESP32]# select-attribute beb5483e-36e1-4688-b7f5-ea07361b26a8
[ESP32:/service0028/char0029]# read
Attempting to read
/org/bluez/hci0/dev_24_0A_C4_11_03_12/service0028/char0029
[CHG] Attribute /org/bluez/hci0/dev_24_0A_C4_11_03_12/service0028/char0029
Value:
  48 65 6c 6c 6f 20 57 6f 72 6c 64 20 73 61 79 73  Hello World says
  20 4e 65 69 6c                                    Neil
```

BLE_scan.ino (1/2)

```
#include <BLEDevice.h>
#include <BLEUtils.h>
#include <BLEScan.h>
#include <BLEAdvertisedDevice.h>

class MyAdvertisedDeviceCallbacks:
  public BLEAdvertisedDeviceCallbacks {
    void onResult(BLEAdvertisedDevice advertisedDevice) {
      Serial.printf("Advertised Device: %s\r\n",
                    advertisedDevice.toString().c_str());
    }
};
```

BLE_scan.ino (2/2)

```
void setup() {
  Serial.begin(115200);
  Serial.println("Scanning...");
  BLEDevice::init("");
  BLEScan* pBLEScan=BLEDevice::getScan();
  pBLEScan->setAdvertisedDeviceCallbacks(new MyAdvertisedDeviceCallbacks());
  pBLEScan->setActiveScan(true);
  BLEScanResults foundDevices=pBLEScan->start(30);
  Serial.printf("Devices found: %d \r\n",foundDevices.getCount());
  Serial.println("Scan done!");
}
void loop() {
  delay(2000);
}
```

```
Scanning...
Advertised Device: Name: ESP32, Address:
24:0a:c4:1d:29:66, serviceUUID: 4fafc201-1fb5-
459e-8fcc-c5c9c331914b, txPower: 3
Devices found: 1
Scan done!
```

BLE_client.ino (1/2)

```
#include <BLEDevice.h>
"4fafc201-1fb5-459e-8fcc-c5c9c331914b"
#define CHARACTERISTIC_UUID "beb5483e-36e1-4688-b7f5-ea07361b26a8"
static BLEUUID serviceUUID("6eb61424-0cb3-4998-9014-9ac6d13c6785");
static BLEUUID    charUUID("55a165ab-e16a-4fb0-87da-1bf1ba93002d");
static BLEAddress *pServerAddress=NULL;
static BLEClient *pClient=NULL;
static BLERemoteCharacteristic *pRemoteCharacteristic=NULL;
static boolean doConnect=false;
static boolean connected=false;
class MyAdvertisedDeviceCallbacks:public BLEAdvertisedDeviceCallbacks {
  void onResult(BLEAdvertisedDevice advertisedDevice) {
    Serial.println("BLE Advertised Device found...");
    Serial.println(advertisedDevice.toString().c_str());
    if (advertisedDevice.haveServiceUUID()
     && advertisedDevice.getServiceUUID().equals(serviceUUID)) {
      Serial.println("Found our device!");
      advertisedDevice.getScan()->stop();
      pServerAddress=new BLEAddress(advertisedDevice.getAddress());
      doConnect=true;
    }
  }
};
```

BLE_client.ino (2/3)

```
bool connectToServer(BLEAddress pAddress) {
    Serial.printf("Forming a connection to %s",pAddress.toString().c_str());
    pClient=BLEDevice::createClient();
    Serial.println(" - Created client");
    pClient->connect(pAddress);
    Serial.println(" - Connected to server");
    BLERemoteService *pRemoteService=pClient->getService(serviceUUID);
    if (pRemoteService==nullptr) { return false; }
    Serial.println(" - Found our service");
    pRemoteCharacteristic=pRemoteService->getCharacteristic(charUUID);
    if (pRemoteCharacteristic==nullptr) { return false; }
    Serial.println(" - Found our characteristic");
    std::string value=pRemoteCharacteristic->readValue();
    Serial.print("The characteristic value was: ");
    Serial.println(value.c_str());
}
```

BLE_client.ino (3/3)

```
void setup() {
  Serial.begin(115200);
  Serial.println("Starting BLE Client application...");
  BLEDevice::init("");
  BLEScan* pBLEScan=BLEDevice::getScan();
  pBLEScan->setAdvertisedDeviceCallbacks(new MyAdvertisedDeviceCallbacks());
  pBLEScan->setActiveScan(true);
  pBLEScan->start(30);
  Serial.println("Scan done!");
}
void loop() {
  if (doConnect) {
    if (connectToServer(*pServerAddress)) {
      Serial.println("We are now connected to the BLE Server.");
      connected=true;
    }
    doConnect=false;
  }
  ...
  delay(2000);
}
```

Résultat client

```
Starting BLE Client application...
BLE Advertised Device found...
Name: cb234, Address: 24:0a:c4:1d:29:66, serviceUUID:
4fafc201-1fb5-459e-8fcc-c5c9c331914b, txPower: 3
Found our device!
Scan done.
Connection to 24:0a:c4:1d:29:66
 - Created client
 - Connected to server
 - Found our service
 - Found our characteristic
The characteristic value was: Hello World says Neil
We are now connected to the BLE Server.
```

BLECharacteristic

- `BLEUUID getUUID();`
- `uint16_t getHandle();`
- `std::string` **`getValue`**`();`
- `void setValue(uint8_t* data,size_t size);`
- `void` **`setValue`**`(std::string value);`
- `void setValue(uint16_t& data16);`
- `void setValue(uint32_t& data32);`
- `void setValue(int& data32);`
- `void setValue(float& data32);`
- `void setValue(double& data64);`
- `void` **`indicate`**`();`
- `void` **`notify`**`();`

BLECharacteristicCallbacks

```
class MyBLECharacteristicCallback:public BLECharacteristicCallbacks {
  void onWrite(BLECharacteristic* pCharacteristic) {
   std::string val=pCharacteristic->getValue();
   Serial.printf("onWrite(): %s\r\n",val.c_str());
  }
  void onRead(BLECharacteristic* pCharacteristic) {
   std::string val=pCharacteristic->getValue();
   Serial.printf("onRead(): %s\r\n",val.c_str());
  }
};
...
BLECharacteristic* pCharacteristic=
     pService->createCharacteristic(...);
pCharacteristic->setCallbacks(
     new MyBLECharacteristicCallback());
```

BLERemoteCharacteristic

- `std::string **readValue**(void);`
- `uint8_t readUInt8(void);`
- `uint16_t readUInt16(void);`
- `uint32_t readUInt32(void);`
- `void writeValue(uint8_t* data,size_t len,bool resp=false);`
- `void **writeValue**(std::string newValue,bool response=false);`
- `void writeValue(uint8_t newValue,bool response=false);`
- `void **registerForNotify**(void (*__notifyCallback__)`
  `(BLERemoteCharacteristic* pBLERemoteCharacteristic,`
  `uint8_t* pData, size_t length, bool isNotify));`

Client notify...

```
static void notifyCallback(
    BLERemoteCharacteristic *pBLERC,
    int8_t *pData,size_t length,bool isNotify) {
  Serial.print("Notify: ");
  //Serial.println(pBLERC->getUUID().toString().c_str());
  char str[length+1];
  for (int i=0;i<length;i++) {str[i]=pData[i];}
  str[length]=0;
  Serial.printf("%s (%d)\r\n",str,length);
}
...
void setup() { /** Scan **/ }
void loop() {
  /** Une fois connecté **/
  pRemoteCharacteristic->registerForNotify(notifyCallback);
  ...
}
```

Modification d'une valeur

- Client : une fois connecté au serveur GATT :

```
unsigned long t=millis()/1000;
String val="Time since boot: "+String(t);
Serial.printf("Value set to: [%s]\r\n",val.c_str());
pRemoteCharacteristic->writeValue(val.c_str());
```

- Serveur :

```
unsigned long t=millis()/1000;
String val="Hello: "+String(t);
Serial.printf("Value set to: [%s]\r\n",val.c_str());
pCharacteristic->setValue(val.c_str());
//pCharacteristic->notify(); // Si un client attend
                             // les notifications
```

Rappel : Appairage BT

- Legacy pairing (avant 2.1)
  - Utilisation d'un code PIN (jusqu'à 16 caractères)
- Secure Simple Pairing (SSP)
  - **Just works** : pas d'interactions !
  - **Numeric comparison** : comparaison d'un nombre de 6 chiffres sur les 2 équipements.
  - **Passkey Entry** : Un code est affiché sur un dispositif est doit être entré sur l'autre.
  - **Out Of Band (OOB)** : Echange d'informations par un autre média (ex. NFC) pendant l'appairage.
    - Pas supporté sur l'ESP32 !

Pairing et Bonding

- **Pairing** (Appairage/Association) : processus d'échange d'informations afin d'établir une connexion chiffrée
- **Bonding** (Création d'un lien): les informations du processus d'appairage sont stockées sur les périphériques, de sorte que le processus d'association ne doit pas être répété à chaque fois que les périphériques se reconnectent.

ESP32 BLESecurity.h (1/3)

```
void setCapability(esp_ble_io_cap_t iocap);
```

- **ESP_IO_CAP_OUT** : DisplayOnly
- **ESP_IO_CAP_IO** : DisplayYesNo
- **ESP_IO_CAP_IN** : KeyboardOnly
- **ESP_IO_CAP_NONE** : NoInputNoOutput
- **ESP_IO_CAP_KBDISP** : Keyboard display

	Display Only	Display Yes/No	Keyboard Only	No Input No Output	Keyboard Display
Display Only	Just Works	Just Works	Passkey Entry	Just Works	Passkey Entry
Display Yes/No	Just Works	Just Works	Passkey Entry	Just Works	Passkey Entry
Keyboard Only	Passkey Entry	Passkey Entry	Passkey Entry	Just Works	Passkey Entry
No Input No Output	Just Works	Just Works	Just Works	Just Works	Just Works
Keyboard Display	Passkey Entry	Passkey Entry	Passkey Entry	Just Works	Passkey Entry

ESP32 BLESecurity.h (2/3)

```
void setAuthenticationMode(esp_ble_auth_req_t auth_req);
```

- **ESP_LE_AUTH_NO_BOND**: No bonding.
- **ESP_LE_AUTH_BOND**: Bonding is performed.
- **ESP_LE_AUTH_REQ_MITM**: MITM Protection is enabled.
- **ESP_LE_AUTH_REQ_SC_ONLY**:
  - Secure Connections without bonding enabled.
- **ESP_LE_AUTH_REQ_SC_BOND**:
  - Secure Connections with bonding enabled.
- **ESP_LE_AUTH_REQ_SC_MITM**:
  - Secure Connections with MITM Protection and no bonding enabled.
- **ESP_LE_AUTH_REQ_SC_MITM_BOND**:
  - Secure Connections with MITM Protection and bonding enabled.

ESP32 BLESecurity.h (3/3)

```
void setKeySize(uint8_t key_size = 16) ;
void setInitEncryptionKey(uint8_t init_key);//e.g. client
void setRespEncryptionKey(uint8_t resp_key);//e.g. serveur
uint8_t init_key=ESP_BLE_ENC_KEY_MASK |
                 ESP_BLE_ID_KEY_MASK;
uint8_t resp_key=ESP_BLE_ENC_KEY_MASK |
                 ESP_BLE_ID_KEY_MASK;
```

ESP32 Security exemple (1/3)

```
class MySecurity:public BLESecurityCallbacks {
  uint32_t onPassKeyRequest(){
    Serial.println("onPassKeyRequest()...");
    return 123456;
  }
  void onPassKeyNotify(uint32_t pass_key){
    Serial.printf("onPassKeyNotify(): %06d\r\n",pass_key);
  }
  bool onConfirmPIN(uint32_t pass_key){
    Serial.printf("onConfirmPIN(): %06d\r\n",pass_key);
    vTaskDelay(5000);
    return true;
  }
  bool onSecurityRequest(){
    Serial.println("onSecurityRequest()...");
    return true;
  }
  void onAuthenticationComplete(esp_ble_auth_cmpl_t cmpl){
    Serial.println("onAuthenticationComplete()...");
  }
};
```

ESP32 Security exemple (2/3)

```
...
BLEDevice::setEncryptionLevel(ESP_BLE_SEC_ENCRYPT);
BLEDevice::setSecurityCallbacks(new MySecurity());
BLESecurity *pSecurity=new BLESecurity();
pSecurity->setCapability(ESP_IO_CAP_OUT);
pSecurity->setAuthenticationMode(ESP_LE_AUTH_REQ_SC_ONLY);
pSecurity->setKeySize();
pSecurity->setInitEncryptionKey(ESP_BLE_ENC_KEY_MASK|
                                ESP_BLE_ID_KEY_MASK);
pSecurity->setRespEncryptionKey(ESP_BLE_ENC_KEY_MASK|
                                ESP_BLE_ID_KEY_MASK);
...
```

ESP32 Security exemple (3/3)

- A ne pas oublier (car c'est pas facile à trouver) !!!

```
int dev_num=esp_ble_get_bond_device_num();
Serial.printf("esp_ble_get_bond_device_num(): %d\r\n",dev_num);
if (dev_num>0) {
  esp_ble_bond_dev_t *dev_list=(esp_ble_bond_dev_t *)
                       malloc(sizeof(esp_ble_bond_dev_t)*dev_num);
  esp_ble_get_bond_device_list(&dev_num,dev_list);
  for (int i=0;i<dev_num;i++) {
    Serial.printf("Dev(%d): ",i);
    for (int j=0;j<ESP_BD_ADDR_LEN;j++) {
      if (j>0) Serial.printf(":");
      Serial.printf("%02x",dev_list[i].bd_addr[j]);
    }
    Serial.printf("\r\n");
    esp_ble_remove_bond_device(dev_list[i].bd_addr);
  }
  free(dev_list);
}
```

Captures de trames BT

```
# tshark -D
1. wlan0
2. lo (Loopback)
3. any
4. eth0
5. docker0
6. bluetooth-monitor
...
11. bluetooth0
...
# tshark -ni bluetooth0
```

Capturing on 'bluetooth0'

1	0.000000	host → controller	HCI_CMD 11 Sent LE Set Scan Parameters
2	0.000775	controller → host	HCI_EVT 7 Rcvd Command Complete (LE Set Scan Parameters)
3	0.000925	host → controller	HCI_CMD 6 Sent LE Set Scan Enable
4	0.002773	controller → host	HCI_EVT 7 Rcvd Command Complete (LE Set Scan Enable)
5	0.602905	controller → host	HCI_EVT 36 Rcvd LE Meta (LE Advertising Report)

...

Références

- https://www.bluetooth.com/specifications/bluetooth-core-specification
- https://www.bluetooth.com/specifications/gatt
  https://www.espressif.com/en/products/hardware/esp32/overview
- https://en.wikipedia.org/wiki/Bluetooth_Low_Energy
- https://github.com/espressif/esp-idf/blob/master/examples/bluetooth/gatt_security_server/tutorial/Gatt_Security_Server_Example_Walkthrough.md