BLUETOOTH LOW ENERGY E-RL LOCK

INTERFACE SPECIFICATION

Version *1.00*

1/09/2016

AXA Bike Security
EnergieStraat 2
3903 AV Veenendaal
Netherlands
http://www.axabikesecurity.com/
T: +31 318 536 111
F: +31 318 595 535

Disclaimer

The information contained in this document is believed to be correct and complete. However, AXA Stenman Nederland B.V. ("AXA") does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information or as to its suitability (for products) and shall have no liability whatsoever for the consequences of use of such information. In no event shall AXA be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

AXA's total liability for any direct damages incurred by the products described herein, if and when established by a competent court, shall be limited to the price of such products.

Customers are responsible for the design and operation of their applications and products using AXA products, and AXA accepts no liability whatsoever for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the AXA product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

AXA reserves the right to modify any of the documentation at any time and without notice. AXA assumes no responsibility for any infringements of patents or other rights of third parties that may result from the use of any product or documentation of AXA.

Copyright

FCC/ISED Regulatory notices

Modification statement

AXA Stenman Nederland B.V. has not approved any changes or modifications to this device by the user. Any changes or modifications could void the user's authority to operate the equipment.

AXA Stenman Nederland B.V. *n'approuve aucune modification apportée à l'appareil par l'utilisateur, quelle qu'en soit la nature. Tout changement ou modification peuvent annuler le droit d'utilisation de l'appareil par l'utilisateur.*

Interference statement (if it is not placed in the device)

This device complies with Part 15 of the FCC Rules and Industry Canada licence-exempt RSS standard(s). Operation is subject to the following two conditions: (1) this device may not cause interference, and (2) this device must accept any interference, including interference that may cause undesired operation of the device.

Le présent appareil est conforme aux CNR d'Industrie Canada applicables aux appareils radio exempts de licence. L'exploitation est autorisée aux deux conditions suivantes : (1) l'appareil ne doit pas produire de brouillage, et (2) l'utilisateur de l'appareil doit accepter tout brouillage radioélectrique subi, même si le brouillage est susceptible d'en compromettre le fonctionnement.

Wireless notice

This device complies with FCC/ISED radiation exposure limits set forth for an uncontrolled environment and meets the FCC radio frequency (RF) Exposure Guidelines and RSS-102 of the ISED radio frequency (RF) Exposure rules. This transmitter must not be co-located or operating in conjunction with any other antenna or transmitter.

Le présent appareil est conforme à l'exposition aux radiations FCC / ISED définies pour un environnement non contrôlé et répond aux directives d'exposition de la fréquence de la FCC radiofréquence (RF) et RSS-102 de la fréquence radio (RF) ISED règles d'exposition. L'émetteur ne doit pas être colocalisé ni fonctionner conjointement avec à autre antenne ou autre émetteur.

FCC Class B digital device notice

This equipment has been tested and found to comply with the limits for a Class B digital device, pursuant to part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference in a residential installation. This equipment generates, uses and can radiate radio frequency energy and, if not installed and used in accordance with the instructions, may cause harmful interference to radio communications. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause harmful interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures:

- Reorient or relocate the receiving antenna.
- Increase the separation between the equipment and receiver.
- Connect the equipment into an outlet on a circuit different from that to which the receiver is connected.
- Consult the dealer or an experienced radio/TV technician for help.

CAN ICES-3 (B) / NMB-3 (B)

This Class B digital apparatus complies with Canadian ICES-003.

Cet appareil numérique de classe B est conforme à la norme canadienne NMB-003.

Table of Contents

1. Introduction

The Bluetooth Low Energy Wireless eRL lock can be controlled directly through standard Bluetooth LE communication without the need for a proprietary communication stack. Bluetooth Low Energy sometimes referred to as Bluetooth 4.X or Bluetooth Smart is a new technology completely different and not compatible with Classic Bluetooth that has been around since 2001. Classic Bluetooth is well known for its use in hands free car kits and wireless earphones remote controls for GSM and smart phones.

The Bluetooth Low Energy (BLE) based eRL takes all complicated lock handling, timing and key management out of the hands of the application (app) builder and offers a simple straightforward standard interface to control and monitor the eRL. The eRL employs an extreme smart low power saving mode minimizing current consumption to maximize battery life while offering selective response to control commands. The eRL remembers and learns the user's behavior and automatically adopts itself to the user's needs maximizing the time it's asleep. The net effect will be that when the user is asleep the eRL will be asleep as well and before the user comes to take his bicycle the eRL will wake up and prepares for just another day at the office. All this is done with only one sole purpose and that is to maximize battery life while minimizing the negative effects for the user.

1.1 Quick Overview of Bluetooth LE

Bluetooth Low Energy (BLE) is a Bluetooth technology which was released into the main Bluetooth standard in 2010 along with the Bluetooth Core Specification Version 4.0. Classic Bluetooth devices had suffered from being extremely energy greedy, and the need for a better energy management was necessary. BLE solved this problem and has since been used on most modern devices (all Apple iPhone since the 4S, Samsung phones since the Galaxy SIII, and many more). BLE allows us to use the bluetooth communication for small data exchanges and hence to avoid consuming as much energy.

1.1.1 Application Perspective

Before we get into the description of how the eRL works in detail, let's simply review a couple characteristics of Bluetooth LE devices and how the eRL fits in these:

- **Servers vs. Clients**: these definitions are fairly classic. The Client emits requests and receives responses while the Server listens for requests and sends responses.
- **Central vs. Peripheral**: The peripheral device has valuable information to share with central devices. The central device treats the received data to fulfill specific tasks.

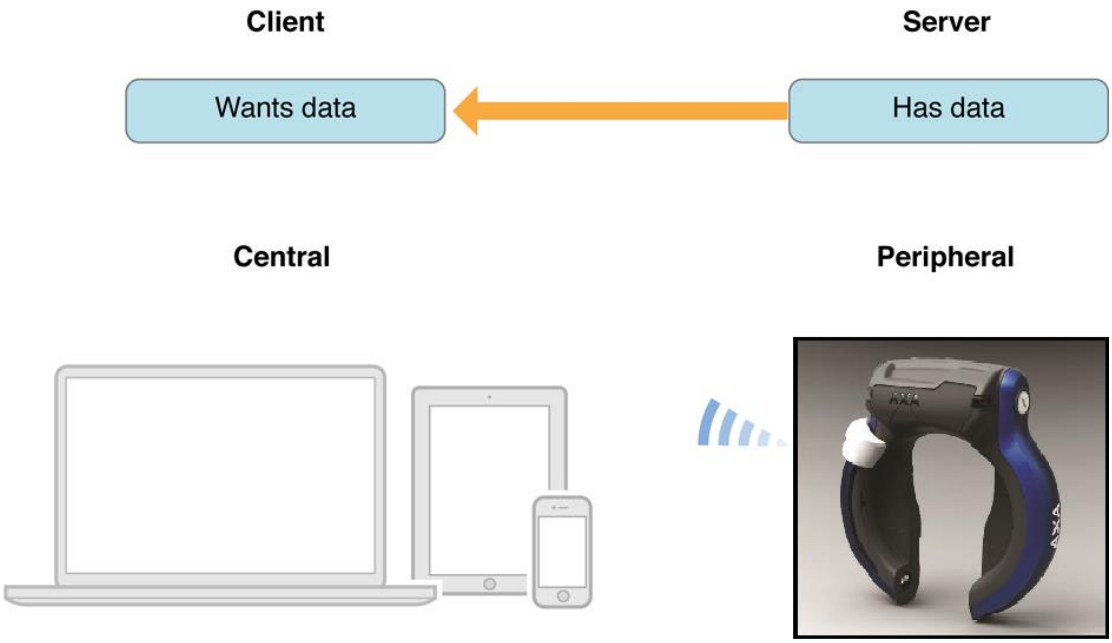

Figure 1.

Figure 1 is giving an overview of the different roles, so what is the difference between a **client** and a **server**? First let me remind you that a client and a server is not interchangeable with master/slave.

Figure 2.

Standard Bluetooth LE peripherals broadcast their presents in order to be found by centrals like smartphone's and tablets, see Figure 2. The eRL is no exception to this, when not connected by a central its broadcasting its presents so other centrals can find it during a scan and make a connection if the peripheral allows connections that is.

Apart from the higher power consumption this introduces a security risk for some product like bicycle locks, thieves will be able to locate bicycles in garages and/or sheds by simply using one of the many freely available Bluetooth LE scanner/localizer apps for smartphones. In order to overcome this kind of problems and to extend the battery lifespan the BLE eRL uses a smart sleeping mechanism, during a sleeping period it's not broadcasting and undetectable even for the bicycle owner's app.

Exiting this particular sleeping state the eRL needs to be woken-up by a subtle movement to start it advertising again during a set period. The smartphone app will now be able to detect the advertising packets and act accordingly depending on the relationship with the eRL.

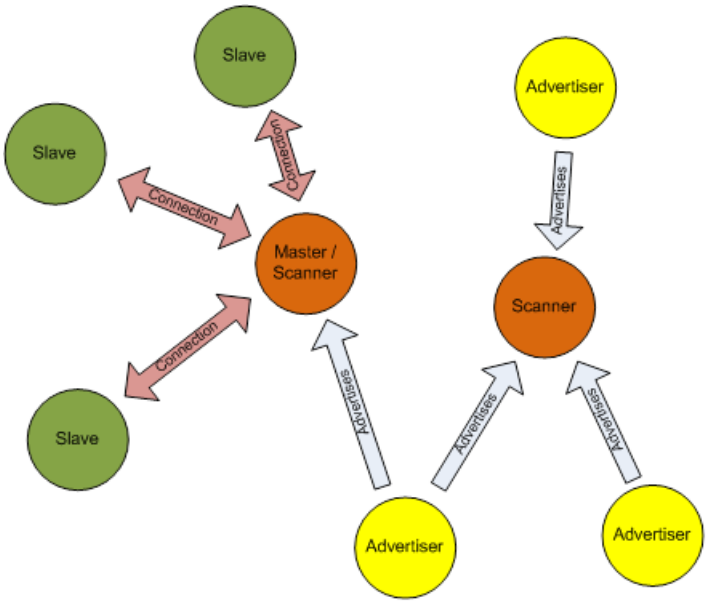

Figure 3.

In real-life situations smartphones (**Central**) will be surrounded by peripherals broadcasting (**Advertising**) their presents, see Figure 3.

A **master (or Central)** is the BLE device that initiates an outgoing connection request to an advertising peripheral device.

A **slave (or Peripheral)** is the BLE device which accepts an incoming connection request after advertising.

A slave can only be connected to one master, but a master can be connected to multiple slaves. In the smartwatch example, your iPhone can theoretically connect to multiple smartwatches at the same time. However, your smartwatch can only ever connect to one smartphone at a time.

There is no limit in the Bluetooth SIG on the number of slaves a master can connect to. Generally this will be limited by the BLE technology or Bluetooth stack you use.

Devices such as smartphones or tablets would generally (but not exclusively) adopt the role of "Scanner" and would "discover" other devices that have adopted the "Advertiser" role by a process called **discovery** which can be **active** ("are there any devices out there?") or **passive** ("I'll listen whilst devices advertise their presence"). Devices that adopt the "Advertiser" role are generally (but not exclusively) smaller footprint devices such as heart rate monitors or temperature sensors.

Once devices have discovered one another one will act as an "**Initiator**" (typically the smartphone or tablet type device) and attempt to connect to one of the devices that it has discovered. If successful it will adopt the role of "**Master**" and the other will adopt the role of "**Slave**". "**Master**" devices will initiate commands and requests to "**Slave**" devices which will respond.

One of the first task of a Bluetooth LE application, such as on a smartphone app, is to discover other Bluetooth LE devices that it can connect to.

1.1.2 Services

Once a device has been discovered the next task is to figure out what services are offered by the device. So, what's a service? Well, a service consists of:

- A Service Specification, which consists of:
  - A collection of characteristics;
  - References to other service.

Figure 4 is giving an visualization to help cement the concept of services and characteristics. When talking about services we use the names:

- **GATT Server**
- **GATT Client**

This highlights the client/server model that is used at this level of the architecture.

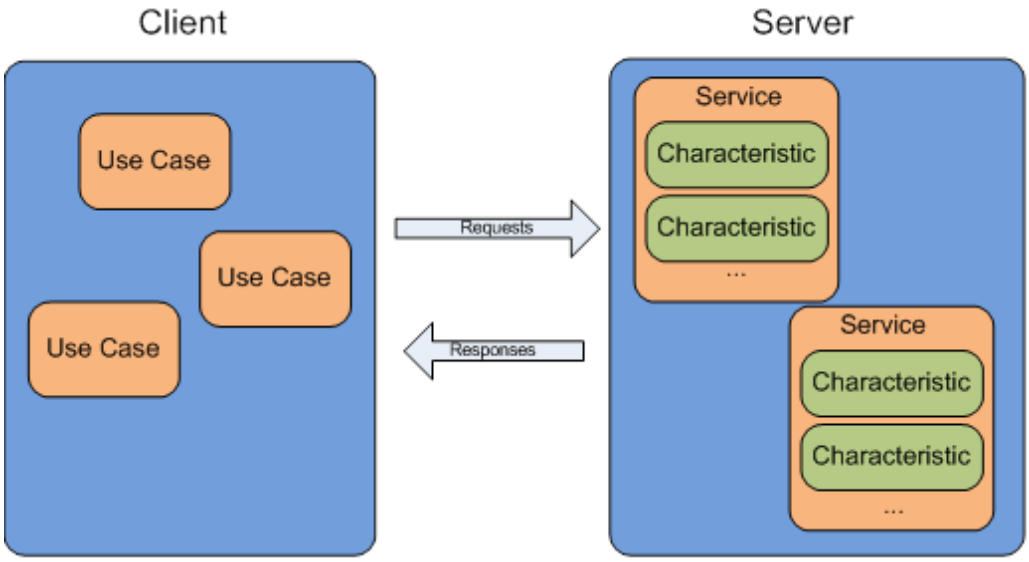

Figure 4.

Let's move on to the differences between a **GATT server** and a **GATT client**

A **GATT client** is a device which accesses data on the remote GATT server via read, write, notify, or indicate operations.

A **GATT server** is a device which stores data locally and provides data access methods to a remote GATT client.

You can easily see that it is possible for a device to be a GATT server and a GATT client at the same time. While it is most common for the slave (peripheral) device to be the GATT server only and the master (central) device to be the GATT client, this is not required. **The GATT functionality of a device is logically separate from the master/slave role.** The master/slave roles control how the BLE radio connection is managed, and the client/server roles are dictated by the storage and flow of data.

1.1.3 Profiles

A GATT database implements one or more **profiles**, and each profile is made up of one or more **services**, and each service is made up of one or more **characteristics**.

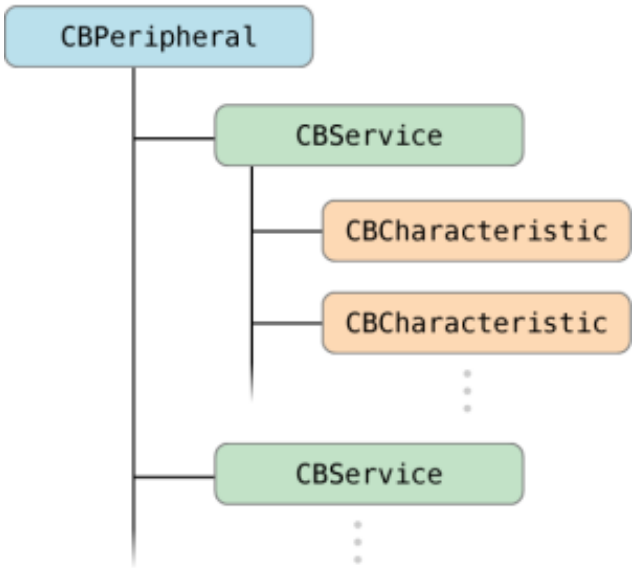

Figure 5.

GATT servers can implement as many profiles, services, and characteristics as needed by the product, figure 5. When using a non-standard profile, a 128bit UUID is required and must be provided by the peripheral producer offering this unique profile. You can also adhere to any **Bluetooth SIG profiles** (services, and characteristics) currently supported, in which case the profile UUID is 16bit long. Every BLE device acting as a GATT server must implement the official **Generic Access service**. This includes two mandatory characteristics: **Device Name** and **Appearance**.

1.1.4 Attributes and Characteristics

Remember that a service is made up of one or more **characteristics**. However, one characteristic may be comprised of many different **attributes**.

Each **attribute** is given a unique numerical **handle** which the GATT client may use to reference, access, and modify it. Every characteristic has one main attribute which allows access to the actual value stored in the database for that characteristic. When you "write a value to a characteristics" or "read the value of the characteristic" you are doing read and write operations on the main data attribute (of said characteristic).

Some other related attributes are read-only (such as a **Characteristic User Description** attribute), some control the behavior of the characteristic (such as the **Client Characteristic Configuration** attribute which is used to enable **notify** or **indicate** operations).

Every attribute has a UUID. These may be either 16 bits (e.g. **"0x180A"**) or 128 bits (e.g. **"00001530-1212-EFDE-1523-785FEABCD123 "**). All 16-bit UUIDs are

defined by the Bluetooth SIG and are known as adopted UUIDs. All 128-bit UUIDs are custom and may be used for any purpose without approval from the Bluetooth SIG. Two very common 16-bit UUIDs that you will see are **0x2901**, the Characteristic User Description attribute and **0x2902**, the Client Characteristic Configuration attribute (to enable either **"notify"** or **"indicate"** on a characteristic).

One important note is that some attribute UUIDs do not technically need to be unique. Their **handles** are always unique, but the UUIDs occasionally overlap. For example, every Client Characteristic Configuration attribute has a UUID of **0x2902**, even though there may be a dozen of them in a single GATT database.

1.1.5 Advertising and Connections

The Generic Access Profile (**GAP**) specifically describes behaviors and procedures for device discovery, connection establishment, security, authentication, and service discovery, and this along with performing a single defining role. In essence, a Bluetooth device may incorporate either initiating or accepting procedures, and the peer device must support the corresponding functionality. One of the most important things to understand with Bluetooth low energy is how two devices first find one another, work out what they can do with one another, and how they can find and connect with one another repeatedly. This is really what GAP defines.

There are four GAP roles defined for a Bluetooth low energy device:

- **Broadcaster**
- **Observer**
- **Peripheral**
- **Central**

A broadcaster is a device that sends advertising packets. Typically, this is used to broadcast some data from a service to other devices that happen to be in an observer role. A broadcast must have a transmitter but does not need a receiver. A broadcast-only device, therefore, only needs a transmitter.

An observer is a device that scans for broadcasters and reports this information to an application. An observer must have a receiver; it can also optionally have a transmitter.

A peripheral is a device that advertises by using connectable advertising packets. As such, it becomes a slave once connected. A peripheral needs to have both a transmitter and a receiver. The eRL has adopted the role of peripheral device and as such is sending advertising packets while not connected.

A central is a device that initiates connections to peripherals. As such, it becomes a master when connected. Just like a peripheral, a central needs to have both a transmitter and a receiver. We explained before that since the eRL has adopted the role of peripheral device the smartphone will need to accept the role of central in this system. The role that is already normal for smartphone's since most other BLE

peripherals connected to the smartphone are peripheral's, for example smartwatches or other body sensors that measure vital signs while cycling. A device can support multiple GAP roles at the same time. For example, a device can be a broadcaster and a peripheral at the same time.

1.1.6 Pairing and Bonding

Just a quick writeup on the difference between pairing and bonding, since these terms get used interchangeably. This has to do with the usage of 'pairing' in Bluetooth Classic, or BR/EDR.

As far as Bluetooth LE is concerned, pairing and bonding are two very distinct things. The short explanations are that pairing is the exchange of security features each device has, and creating temporary encryption for the livecycle of the connection. Bonding is the exchange of long term keys **after pairing has occurred**, and **storing those keys for later use**. Pairing is not the creation of permanent security between devices, that is called bonding. Pairing is the mechanism that allows bonding to occur.

Pairing

Pairing is the exchange of security features. This includes things like i/o capabilities, requirement for man-in-the-middle protection, etc. The client side begins this exchange. The client essentially says 'hey, i'd like it if you had these features'. The server replies, 'yeah, well, this is what I can do'. Once this exchange is made, the security that will be used has been determed. For example, if a server supports just noInput/noOutput for i/o capabilities, the Just Works pairing mechanism is going to be used. Once the pairing feature exchange is complete, a temporary security key is exchanged and the connection is encrypted, but only using the temporary key. In this encrypted connection, long term keys are exchanged. These keys are things like the (long term) encryption key to encrypt a connection, and also things like a digital signature key. The exact keys exchanged are determined by the security features of each device.

Bonding

This really just means that after the pairing features exchange and the connection has been encrypted (these two together are called 'pairing'), and keys have been exchanged, the devices store and use those keys the next time they connect. Keys can be exchanged using the bonding procedure, but that does not mean they are bonded if the keys are not stored and used the next time.

If a device is bonded with another device, like a heart rate monitor and a smartphone, they can encrypt the connection without exchanging any sensitive security information. When the smartphone connects to the heart rate monitor, it can just issue a 'turn on encryption' request, and both sides will use the keys already stored, so nobody snooping can see a key exchange and therefore decode the messages being sent, as is done when pairing.

Certificate

Standard BLE does not use certificates for setting up a secure connection between the master and slave, the eRL does however use certificates signed by the cloud certificate authority KeySafe for setting up secure connections or creating secure relationships. Without the certificate handover by the master (e.g. smartphone) and positive outcome of the verification the eRL will not allow any device to pair, all pairing requestes by the master will be rejected with an insufficient authentication error code. When the certificate is accepted by the eRL it will initiate the setup of a secure link between the devices and allows the user to input the 6 digit passkey on older smartphones or on the more modern smartphones allows the app to input the 128 bit passkey for the user automatically. Entering the passkey is only required once during pairing and bonding, all other times the smartphone will remember the saved bonding information and connects flawlessly with the eRL until the certificates time has expired. Both the certificate and passkey are provided by the KeySafe cloud service upon requesting for an eKey. Next to the standard pairing and bonding the eRL also does offer a secure relationship based on OTP (one-time-pass) commands. Using OTP's does not require the mobile device to pair and bond to securely command the lock to operate. The advantage is trivial, no need to enter passkeys or solving the many problems related to pairing and bonding in many iOS and Android smartphones due to software bugs in the smartphones BLE stack or software version specific particularities. Both systems will be completely compatible with the future revisions of BLE which will introduce **Diffie–Hellman key exchange** to secure the pairing and bonding even further.

1.1.7 Radio Communication

Classic and Bluetooth LE operates in the 2.45 GHz band which it shares with Wi-Fi, Zigbee and microwave ovens worldwide!. Bluetooth LE still retains its fundamental resilience by splitting its radio traffic across 40 channels as shown below (Figure 6).

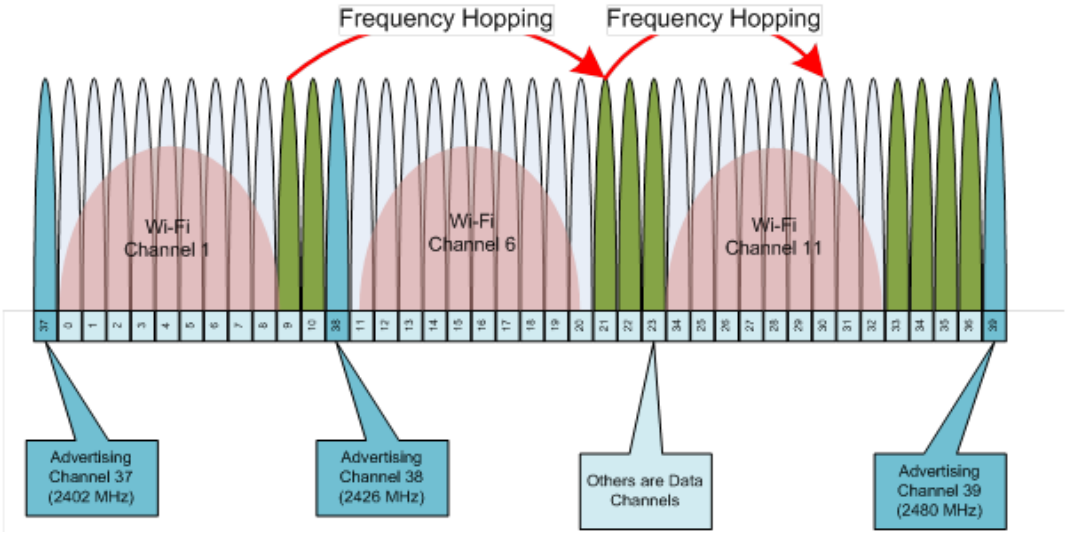

Figure 6.

The Bluetooth low energy channels differ from classic channels because of the relaxed modulation index. This means that the radio energy for each channel is spread wider; therefore, to prevent interference between adjacent Bluetooth low-energy channels, they are separated by 2MHz, instead of the classic 1MHz. In the Link Layer, these channels are divided into two types: advertising channels and data channels. These channel types are aligned with the advertising packets and data packets, as described earlier. When a packet is transmitted, if the packet is sent on an advertising channel, it is an advertising packet. If the packet is sent on a data channel, it is a data packet. There are 3 advertising channels and 37 data channels, as shown in Figure 6 (the advertising channels are rendered in light blue shading). The 3 advertising channels are not all placed in the same part of the ISM band because that would mean that any deep fade in a single part of the band would stop all advertising. Instead, the advertising channels are placed a minimum of 24MHz apart from one another.

The advertising channels are placed strategically away from significant interferers such as a Wi-Fi access point. These public access points typically use one of three 802.11 channels, either channel 1, channel 6, or channel 11. These channels have center frequencies of 2412MHz, 2437MHz, and 2462MHz and a width of approximately 20MHz. This means that channel 1 extends from 2402MHz to 2422MHz, channel 6 extends from 2427MHz to 2447MHz, and channel 11 extends from 2452MHz to 2472MHz. The advertising channels are placed at 2402MHz, 2426MHz, and 2480MHz. This means that the first advertising channel is below Wi-Fi channel 1, the second advertising channel is between Wi-Fi channel 1 and channel 6, and the third advertising channel is above Wi-Fi channel 11. This is illustrated in Figure 6, in which 3 Wi-Fi channels have blocked the use of data channels 0 to 8, 11 to 20, and 34 to 32. The 3 advertising channels, 37, 38, and 39, are all interference free. Bluetooth LE Radio traffic hops around these channels in a pseudo random manner so that the data with get through even though it's in an areas shared by a number of Wi-Fi networks, or microwave ovens. One of the differences between Bluetooth LE and classic Bluetooth is the number and use of these channels.

When in a data connection, an adaptive frequency-hopping algorithm is used. Adaptive frequency hopping makes it possible for a given packet to be remapped from a known bad channel to a known good channel so that the interference from other devices is reduced. To do this, a channel map of good and bad channels is kept in both devices. If the channel that would have been chosen by the master device is a good channel, then that channel is used; if the channel that would have been chosen is a bad channel, then it is remapped onto the set of good channels. A minimum of two data channels must be marked as good by a master.

Suppose, for example, that a Bluetooth low energy device is in the same area as a Wi-Fi channel 1 access point that is streaming data to another Wi-Fi device. The Bluetooth low energy device would mark Link Layer data channels 0 to 8 as bad channels. This means that when the two devices are communicating, they would cycle through the channels and remap these channels to a set of good channels,

1.2 System Description

Making the complicated system setup of the trusted third party KeySafe cloud service and its operation clear a visualization in Figure 7 is helping cement the concept of the KeySafe cloud service. The renting app on the smartphone is detecting the presents of bikes available for renting and offering this to the customer. The customer is making his or her selection and the app is sending renting details to the renting companies' backend computer system for availability. When approved by the renting agent backend computer system an specific eKey type will be requested for this particular bike from the KeySafe, the cloud computer is generating a certificate holding information regarding the bike, renting time and type of eKey. The KeySafe cloud will forward this information including a freshly generated passkey in case of pairing and bonding eKey type to the requesting rental agent backend computer who will forward this information to the renting app.

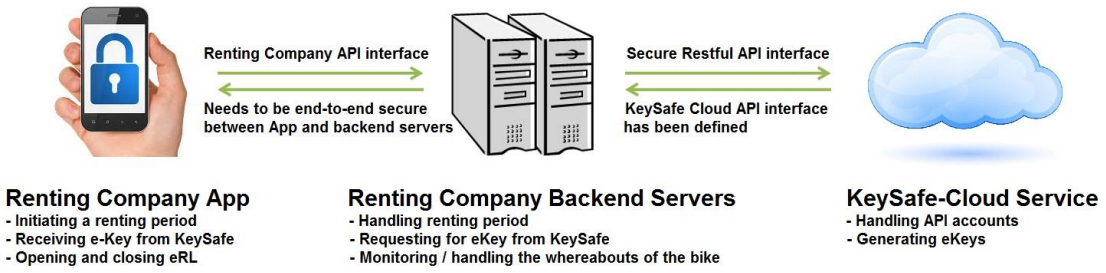

Figure 7.

It's up to the renting app to present the certificate to the eRL BLE service on the selected bike for verification and approval and send the passkey using an API call to the OS on the smartphone. When the certificate is accepted by the eRL and depending on the type of eKey a secure pairing/bonding sequence will follow providing a permanent bond during the specified renting time period. When the renting period has expired it will not automatically close the eRL but will not allow the renting customer to unlock the Bike anymore. The permanent bond with the smartphone is not transferable to a different smartphone since its using unique session keys exchanged during the pairing and bonding process.

In case an OTP eKey certificate is requested from the KeySafe cloud the eKey is returned including a list of OTP commands used to lock and unlock, each of these OTP commands can only be used once and are transferable and not bound to the smartphone that initially send the eKey certificate. OTP eKey certificate are mainly but not exclusively intended for smartphones that do not allow the app to enter the passkey through an API call to the OS.

In Chapter 3 a more detailed and thorough explanation of the different eKey types and there working is given.

2. e-RL Profile

The eRL profile is offering the following 6 Standardised and non-standard services:

- The Standardised Service, which consists of:
  - **Generic Access Service**. (Mandatory for all BLE devices)
  - **Generic Attribute Service**. (Mandatory for all BLE devices)
  - **Battery Service**.
  - **Device Information Service**.

- The Non-standard Service, which consists of:
  - **Lock Command Control Service**.
  - **Device Firmware Update Service**.

All services are explained in the following sections and furthermore specified in detail in the Appendices of this document.

2.1 Battery Service

The Battery Service is a standardized Bluetooth service that exposes the Battery State and Battery Level characteristic (as defined in **Appendix A**) of the CR123A (CR17335) primary lithium battery in the eRL lock. The Battery Level characteristic returns the current battery level as a percentage from 0% to 100%; 0% represent a battery that is fully discharged, 100% represents a battery that is fully charged.

2.2 Device Information Service

The Device Information Service is a standardized Bluetooth service that exposes the eRL device specific information characteristics (as defined in **Appendix B**) including the following version specific information:

- Manufacturer Name String:
  - “Axa Stenman Nederland BV”
- Model Number String:
  - “eRL 2016 B1”
- Serial Number String:
  - “2EA0A-5780B-AD92E-3FEDC”
- Hardware Revision String:
  - “V1.10”

- Software Revision String:
  - “V1.00”
- Firmware Revision String:
  - “V8.00”

This information shall be used by the client application (e.g. app) on how to proceed with the connection and which services and features can be expected from the eRL. The serial number string contains the number which relates to the KeySafe cloud server, the number does have a different format compared to the same number broadcasted by advertisements to make it more and better readable for humans. The hardware revision string is relating to the BLE PCB inside the eRL only and is used together with the software and firmware revision strings during the device firmware update process.

2.3 Lock Command Control Service

The device specific eRL Lock Command Control Service is a non-standard Bluetooth service that exposes an interface for the operation and control of the eRL. In Chapter 3 a more detailed and thorough explanation of the operation of this vital service is given.

2.4 Device Firmware Update Service

The device specific eRL Device Firmware Update Service is a non-standard Bluetooth service that exposes a secure interface for FOTA. Currently a special app is required to update the internal software, the app will upload the new software and/or firmware after which the eRL will be update and ready again for use. This updating process takes less than 1 minute and is secured through the KeySafe-cloud.

3. Operation

Bluetooth Low Energy works with standard profiles and custom profiles every profile can have a number of characteristics as explained in chapter 1, attributes can have different characteristics like read only, write only, notify and many others not relevant at this moment. Standard GATT based profiles, like the Battery and Proximity profiles and many more have a universally unique identifier (UUID) of 16 bits, custom UUID's are 128 Bit long. All UUID's that are custom and not standardised by the Bluetooth SIG have a length of 128 Bits, the Axa UUID used in the eRL is no exception to this since no standard Bluetooth SIG profile and hence UUID has been defined for Locks and in particular Bicycle Locks. The Axa Base UUID used for the eRL is:

0000XXXX-E513-11E5-9260-0002A5D5C51B

Figure 8. – Axa Base UUID

All BLE Axa eRL Locks have the same UUID and hence follow the same custom profile specification. The profile currently holds two attributes, one attribute is used to control, opening/closing and uploading the eKey of the lock while the other attribute is to read the current lock state. In the future more attributes can be defined and they all will use the Axa base UUID. The following attributes are used by the eRL Lock:

- UUID_SERVICE **0x1523**
- UUID_LOCK_CHAR **0x1525** **(Lock Control)**
- UUID_STATE_CHAR **0x1524** **(Lock Status)**

Converting the Axa Base UUID to the particular attribute is simple, simply put the 4 digits from the attribute listed above in the 4 positions underlined in Figure 8 the result can be seen below.

- UUID_SERVICE **00001523-E513-11E5-9260-0002A5D5C51B**
- UUID_LOCK_CHAR **00001525-E513-11E5-9260-0002A5D5C51B**
- UUID_STATE_CHAR **00001524-E513-11E5-9260-0002A5D5C51B**

The process of discovering, connecting to and operating the eRL is constructed of the following steps which will be explained in detail in the following sections:

- Discovery / Scanning
- Connecting
- eKey Exchange
- Pairing and bonding (Optional)
- Command and control

3.1 Discovering the e-RL

The Generic Access Profile (GAP) is the cornerstone that allows Bluetooth LE devices to interoperate with each other. It provides a framework that any BLE implementation must follow to allow devices to discover each other, broadcast data, establish secure connections, and perform many other fundamental operations in a standard, universally understood manner. To identify and connect the eRL is advertising using GAP with a fixed advertisement interval of currently 1 second.

During this advertising period it's the app's responsibility to identify the eRL and connect to it if the UID matches. In the advertising packet every eRL lock will send a marker that is indicating the presents of additional scanning information and the Axa UUID to filter only for eRL locks, this feature will keep the main advertising indication as small as possible and provides easy scanning information for app's searching for a specific eRL lock from a group of maybe hundreds of locks.

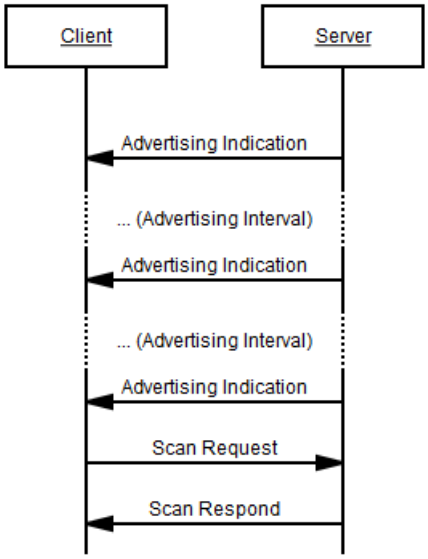

Figure 9. – Scanning for the eRL

Figure 9 is showing the scanning process following the advertising indication send by the eRL during a broadcast. The Scan Request is a standard BLE API call which will be followed by a Scan Respond message from the eRL holding the Serial Number String with the UID. It is advised to use the filter capabilities of the platform if available to filter first on the Axa UUID during scanning and later by filtering the UID inside the app. The UID as formatted in Figure 10 can now be used to check whether this particular eRL is the one we are looking for. It is advised to check locks with the highest signal strength first since its more likely that customers are standing next to the intended bike when scanning for an eRL lock, this is only needed if filtering is a time consuming process on the smartphone. Most of the time this is not the case.

AXA:2EA0A5780BAD92E3FEDC

Figure 10. – An example eRL UID Scan Respond message

3.2 Connecting to the e-RL

During the initial discovery process of the eRL the UID can be extracted, following the Scan Respond message the UID can be matched with the wanted eRL UID if they match the initial connection process from figure 11 shall be initiated.

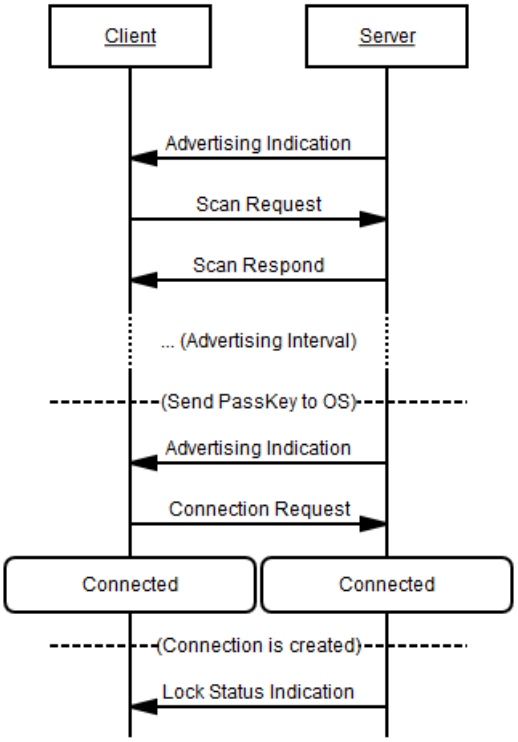

Figure 11. – Initial Connection

Prior to the connection the decision needs to be made what kind of relationship the smartphone will have with the eRL. There are two types of relationships, one is a bond establishing a secure connection and exchanging long term keys the other is no bond.

eKey Type:	Bonded/ Secure Connection:
OTP eKey	No
6 Digit eKey	Yes
128 Bit eKey	Yes
Inv. eKey	No

For long term rental periods the bonded relationship has an advantage however for short term rental periods the OTP eKey has a clear advantage since the initial renting process is faster and no passkey needs to be provided to the OS. In iOS it's not possible to send a passkey by software and therefor the user will have to enter a 6 digit code, in Android it's possible to enter a 6 digit code in most versions by API call to the OS however still a popup window may appear which needs to be suppressed by the app designer or ignored by the user.

If the eRL is known to the smartphone and bonding information has been exchanged during the initial connection process the re-connection process from Figure 12 should be followed to re-establish a prior secure connection, most of the time this is done completely automatic by the smartphone.

Depending on the situation and platform the passkey has to be send to the OS prior to the connection or sometime after the connection has been established, this is highly platform dependent and out of the scope of this document.

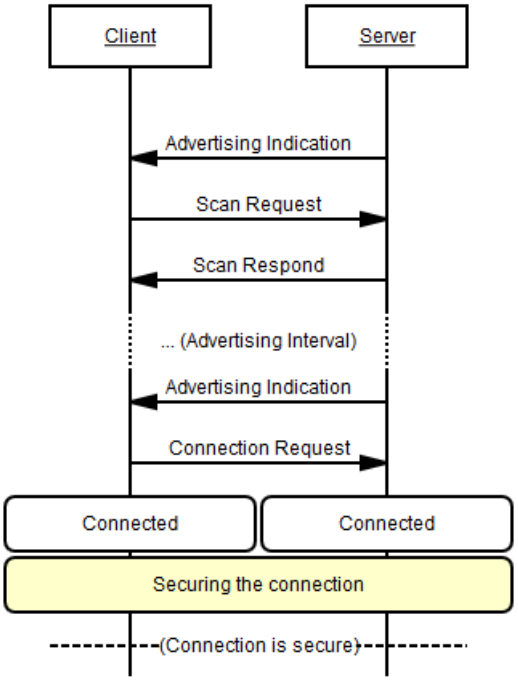

Figure 12. – Re-Connection

3.3 eKey Exchange

Prior to the renting period is started the client device needs to upload the eKey certificate provided by the KeySafe cloud as can be seen in Figure 13.

Depending on the BLE versions supported in the client and server this is either the segmented eKey or the completely unsegmented eKey, before this can be uploaded the eKey needs to be converted from an ASCII HEX string to a Binary string format. This conversion process will effectively halve the length of the string prior to transmission.

The segmented eKey received from the KeySafe cloud should be segmented on the “-” dash points, resulting in 6 strings of which 5 will be 40 characters and one string of 20 characters long. The conversion from an ASCII HEX string to a Binary string format will create 5 strings of 20 bytes and one string of 10 bytes long which shall be send to the eRL to initiate a renting period. When the eKey certificate is accepted by the eRL depending on the eKey type it will initiate the security request starting the whole bonding-pairing process automatically.

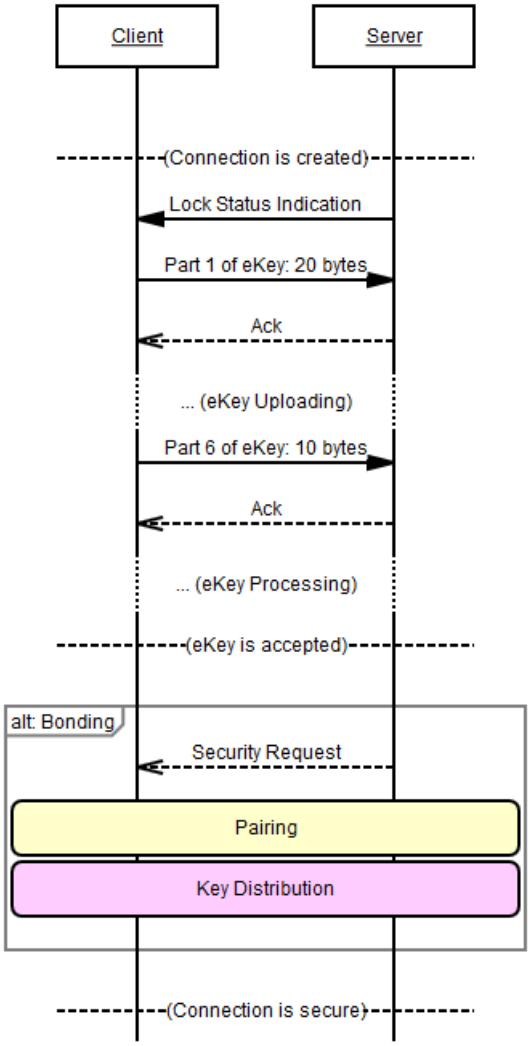

Figure 13. – Uploading the eKey

After receiving the first valid eKey and successful acceptance the eRL will be out of virgin mode and into normal mode, only accepting OPEN/CLOSE commands during a valid secure connection started with an eKey. The eKey is only needed for setting-up the connection, the eKey can only be used once and can be discarded after setting-up the connection since it's no longer required to re-establish subsequent connections.

The different eKey types have different functions, the OTP eKey does not create a secure bond but a secure relationship with the OTP commands received from the cloud. These commands can be used to open and close the eRL, these commands need to be used in sequence. The last OTP command is always a close command and will also end the renting contract. If you want the renting contract to be ended early you can jump directly to the last OTP command but always need to send the OTP command before the last one first. You basically will send two OTP commands in this case, the one before the final and the final.

The Inv. eKey is a special case, this eKey is the invalidate eKey and will end any renting contact without starting a new contract and changing the status of the lock. This eKey can be used in a secure bonded connection or a secure relationship. The two secure bonded eKeys, 6 Digit and 128 Bit eKey are creating a secure bonded connection with the smartphone whereby all communication is encrypted. This bond can be permanent or time limited, this is specified at the start of the bonding process.

3.4 Bonding-Pairing to the e-RL

The pairing algorithm used by the eRL protects against MITM attacks and is currently the most secure pairing process available in V4.0 and V4.1. The initial bonding-pairing process will create key-pairs that are saved on both peer devices will be used to re-establish a secure connection the next time a connection request is initiated. The bonding-pairing process should only be performed once during the initial connection process to setup the secure connection using the eKey. All subsequent connections will reuse the keys exchanged between the server/client to re-establish a secure connection. Prior to the bonding process is started the client device needs to upload the eKey certificate provided by the KeySafe cloud as can be seen in Figure 13. It's the app's responsibility to discard the bonding-pairing information on the smartphone after a renting-period has ended or prematurely is terminated.

3.5 Command and Control

Controlling the eRL is done through the UUID_LOCK_CHAR and the lock status is available through the UUID_STATE_CHAR characteristic. Depending on the type of eKey used a different type of command needs to be given. OTP eKey requires the transmission of 20 bytes OTP commands to change the lock, the eKey that creates a secure bonded connection needs to send a one byte command to open or close the lock. The eRL lock does have different modes and below they are explained in detail.

Unlocked Mode – Secured / Unsecured / Unexpected

Figure 14 presents the condition the leaver is in when the lock is Unlocked and the leaver is secured. This is the normal condition the lock is in when Unlocked.

Figure 14. – Unlocked


Figure 15. – Unsecured


Figure 16. – Locked

The status reported to the user will be “Unlocked”. Figure 15 presents the condition the leaver is in when the lock is Unlocked and the leaver is unsecured and is still able to move. This condition happens when the user Unlocks the lock and the spring inside the lock is not able to move the leaver completely in the Unlocked condition of Figure 14 due to an object (eq. spokes, hand) blocking this. This is an abnormal condition since the lock can now either be put in the Unlocked secured leaver condition from Figure 14 or in the Locked condition from Figure 16. The status reported will be Unlocked in both the Unlocked secured (Fig 14) and Unlocked unsecured conditions (Fig 15), if the leaver is moved from the Unlocked unsecured into the Locked condition from Figure 16 the status reported will be updated and reports Locked.

Bluetooth LE Mode – Unlocking / Locking

The Unlocking / Locking sequence is controlled by the attribute property UUID_LOCK_CHAR sending a 0x00 will open the eRL Lock when closed and sending a 0x01 will close the eRL Lock when open for secure bonded connections, secure relationship will require the sending of an OTP command. The status attribute property UUID_STATE_CHAR will reflect the current eRL Lock status, 0x01 will represent “closed” and 0x00 will represent “open”.

LOCK_OPEN_STATUS	0x00	
LOCK_UNSECURED_OPEN_STATUS	0x08	(child safety removed)
LOCK_WEAK_CLOSED_STATUS	0x09	
LOCK_STRONG_CLOSED_STATUS	0x01	
LOCK_ERROR_STATUS	0xFF	

Status values larger than 0x0F and smaller than 0xFF should be ignored since they are reserved for extensions. All status reports are one byte this means that multi-byte reports should be ignored too.

4. Test Scenarios

Test scenarios are test cases designed for the software designers that ensure that all possible situations and conditions are tested from end to end. The scenarios are independent tests; or a series of tests that follow each other, where each of them depends upon the output of the previous one. The scenarios were prepared by reviewing general functional requirements, and are designed to represent both typical and unusual situations that may occur in the user case. The test scenarios are executed through the use of test procedures given in Appendix F, the procedures define a series of steps necessary to perform one or more test scenarios.

5. Electrical Characteristics

Absolute Maximum Electrical Ratings(†)

Characteristic	
Ambient temperature under bias	-25°C to +85°C
Ambient temperature during operation	-20°C to +55°C
Supply Voltage Vdd pin with respect to Vss (Idle)	3V
Supply Voltage Vdd pin with respect to Vss (Running)	3V
Supply Voltage Vdd pin with respect to Vss (Stall)	3V
Supply Current Vdd pin	250mA
Peak (10s) Supply Current Vdd pin @ 3V (Stall)	500mA
ESD protection on all pins (HBM; MM)	≥ 2kV; ≥ 400V

Standard Electrical Operating Conditions (unless otherwise stated)

Operating temperature -20°C ≤ TA ≤ +55°C

Characteristic	Min	Typ.	Max	units	Conditions
Supply Voltage (Vdd)	2.2	3	3.6	V	Normal Mode
Supply Voltage (Vdd)	2.8	3	3.6	V	BLE FOTA Mode
Supply Current		90		mA	Motor Running
Peak Supply Current			350	mA	Motor Startup < 100ms
Motor Stall Current			500	mA	Motor Stall < 10s @ 3V
Power-saving Current[1]		7	15	µA	Vdd = 3V, (Sleep mode)
Start-up Time		290	500	ms	Power up
Unlocking Time	1.5	1.8	2.1	s	Locked -> Unlocked
Locking Timeout	14.9	15	15.1	s	Unlocked -> Unlocked
Stall Timeout	9.9	10	10.1	s	Vdd = 3V

Appendix A – Battery Service

Name: Battery Service

Type: org.bluetooth.service.battery_service
Assigned Number: 0x180F

Abstract:
The Battery Service exposes the state of a battery within the eRL lock.

Summary:
The Battery Service exposes the Battery State and Battery Level of the CR123A (CR17335) primary lithium single battery in the eRL lock.

Service Dependencies
This service has no dependencies on other GATT-based services.

GATT Requirements

Sub-Procedure	Server Requirement
Read Characteristic Descriptors	Mandatory
Notifications	C1: Mandatory if the Battery Level characteristic properties supports notification, otherwise excluded.
Write Characteristic Descriptors	C1: Mandatory if the Battery Level characteristic properties supports notification, otherwise excluded.

Transport Dependencies

Transport	Supported
Classic	true
Low Energy	true
High Speed	

Error Codes

This service does not define any application error codes that are used in Attribute Protocol.

Service Characteristics

<table>
<tr><th>Overview</th><th>Properties</th><th>Security</th><th>Descriptors</th></tr>
<tr>
<td>Name:
Battery Level
Description:
The Battery Level characteristic is read using the GATT Read Characteristic Value sub-procedure and returns the current battery level as a percentage from 0% to 100%; 0% represents a battery that is fully discharged, 100% represents a battery that is fully charged.
Type:
characteristic.battery_level
Requirement:
Mandatory</td>
<td>
<table>
<tr><th>Property</th><th>Requirement</th></tr>
<tr><td>Read</td><td>Mandatory</td></tr>
<tr><td>Write</td><td>Excluded</td></tr>
<tr><td>WriteWithoutResponse</td><td>Excluded</td></tr>
<tr><td>SignedWrite</td><td>Excluded</td></tr>
<tr><td>Notify</td><td>Optional</td></tr>
<tr><td>Indicate</td><td>Excluded</td></tr>
<tr><td>WritableAuxiliaries</td><td>Excluded</td></tr>
<tr><td>Broadcast</td><td>Excluded</td></tr>
<tr><td>ExtendedProperties</td><td></td></tr>
</table>
</td>
<td>None</td>
<td>
<table>
<tr><th>Overview</th><th>Permissions</th></tr>
<tr>
<td>Name:
Characteristic Presentation Format
Type:
descriptor.gatt.characteristic_presentation_format
Requirement:
if_multiple_service_instances</td>
<td>
<table>
<tr><th>Permission</th><th>Requirement</th></tr>
<tr><td>Read</td><td>Mandatory</td></tr>
<tr><td>Write</td><td>Excluded</td></tr>
</table>
</td>
</tr>
<tr>
<td>Name:
Client Characteristic Configuration
Type:
descriptor.gatt.client_characteristic_configuration
Requirement:
if_notify_or_indicate_supported</td>
<td>
<table>
<tr><th>Permission</th><th>Requirement</th></tr>
<tr><td>Read</td><td>Mandatory</td></tr>
<tr><td>Write</td><td>Mandatory</td></tr>
</table>
</td>
</tr>
</table>
</td>
</tr>
</table>

Appendix B – Device Information Service

Name: Device Information

Type: org.bluetooth.service.device_informationDownload / View
Assigned Number: 0x180A

Abstract:
The Device Information Service exposes manufacturer and/or vendor information about a device.

Summary:
This service exposes manufacturer information about a device. The Device Information Service is instantiated as a Primary Service. Only one instance of the Device Information Service is exposed on a device.

Service Dependencies
This service is not dependent upon any other services.

Transport Dependencies

Transport	Supported
Classic	true
Low Energy	true
High Speed	

Error Codes
This service does not define any application error codes that are used in Attribute Protocol.

Service Characteristics

Overview	Properties	Security	Descriptors
Name: Manufacturer Name String **Description:** This characteristic represents the name of the manufacturer of the device. **Type:** org.bluetooth.characteristic.manufacturer_name_string **Requirement:** Optional	**Property** / **Requirement** Read: Mandatory Write: Excluded WriteWithoutResponse: Excluded SignedWrite: Excluded Notify: Excluded Indicate: Excluded WritableAuxiliaries: Excluded Broadcast: Excluded ExtendedProperties:	None	None
Name: Model Number String **Description:** This characteristic represents the model number that is assigned by the device vendor. **Type:** org.bluetooth.characteristic.model_number_string **Requirement:** Optional	**Property** / **Requirement** Read: Mandatory Write: Excluded WriteWithoutResponse: Excluded SignedWrite: Excluded Notify: Excluded Indicate: Excluded WritableAuxiliaries: Excluded Broadcast: Excluded ExtendedProperties:	None	None

<table>
<tr><th>Overview</th><th>Properties</th><th>Security</th><th>Descriptors</th></tr>
<tr>
<td>Name:
Serial Number String
Description:
This characteristic represents the serial number for a particular instance of the device.
Type:
org.bluetooth.characteristic.serial_number_string
Requirement:
Optional</td>
<td><table>
<tr><th>Property</th><th>Requirement</th></tr>
<tr><td>Read</td><td>Mandatory</td></tr>
<tr><td>Write</td><td>Excluded</td></tr>
<tr><td>WriteWithoutResponse</td><td>Excluded</td></tr>
<tr><td>SignedWrite</td><td>Excluded</td></tr>
<tr><td>Notify</td><td>Excluded</td></tr>
<tr><td>Indicate</td><td>Excluded</td></tr>
<tr><td>WritableAuxiliaries</td><td>Excluded</td></tr>
<tr><td>Broadcast</td><td>Excluded</td></tr>
<tr><td>ExtendedProperties</td><td></td></tr>
</table></td>
<td>None</td>
<td>None</td>
</tr>
<tr><td></td><td></td><td></td><td></td></tr>
<tr>
<td>Name:
Hardware Revision String
Description:
This characteristic represents the hardware revision for the hardware within the device.
Type:
org.bluetooth.characteristic.hardware_revision_string
Requirement:
Optional</td>
<td><table>
<tr><th>Property</th><th>Requirement</th></tr>
<tr><td>Read</td><td>Mandatory</td></tr>
<tr><td>Write</td><td>Excluded</td></tr>
<tr><td>WriteWithoutResponse</td><td>Excluded</td></tr>
<tr><td>SignedWrite</td><td>Excluded</td></tr>
<tr><td>Notify</td><td>Excluded</td></tr>
<tr><td>Indicate</td><td>Excluded</td></tr>
<tr><td>WritableAuxiliaries</td><td>Excluded</td></tr>
<tr><td>Broadcast</td><td>Excluded</td></tr>
<tr><td>ExtendedProperties</td><td></td></tr>
</table></td>
<td>None</td>
<td>None</td>
</tr>
<tr><td></td><td></td><td></td><td></td></tr>
</table>

<table>
<tr><th>Overview</th><th>Properties</th><th>Security</th><th>Descriptors</th></tr>
<tr><td>Name:
Firmware Revision String
Description:
This characteristic represents the firmware revision for the firmware within the device.
Type:
org.bluetooth.characteristic.firmware_revision_string
Requirement:
Optional</td><td><table><tr><th>Property</th><th>Requirement</th></tr><tr><td>Read</td><td>Mandatory</td></tr><tr><td>Write</td><td>Excluded</td></tr><tr><td>WriteWithoutResponse</td><td>Excluded</td></tr><tr><td>SignedWrite</td><td>Excluded</td></tr><tr><td>Notify</td><td>Excluded</td></tr><tr><td>Indicate</td><td>Excluded</td></tr><tr><td>WritableAuxiliaries</td><td>Excluded</td></tr><tr><td>Broadcast</td><td>Excluded</td></tr><tr><td>ExtendedProperties</td><td></td></tr></table></td><td>None</td><td>None</td></tr>
<tr><td></td><td></td><td></td><td></td></tr>
<tr><td>Name:
Software Revision String
Description:
This characteristic represents the software revision for the software within the device.
Type:
org.bluetooth.characteristic.software_revision_string
Requirement:
Optional</td><td><table><tr><th>Property</th><th>Requirement</th></tr><tr><td>Read</td><td>Mandatory</td></tr><tr><td>Write</td><td>Excluded</td></tr><tr><td>WriteWithoutResponse</td><td>Excluded</td></tr><tr><td>SignedWrite</td><td>Excluded</td></tr><tr><td>Notify</td><td>Excluded</td></tr><tr><td>Indicate</td><td>Excluded</td></tr><tr><td>WritableAuxiliaries</td><td>Excluded</td></tr><tr><td>Broadcast</td><td>Excluded</td></tr><tr><td>ExtendedProperties</td><td></td></tr></table></td><td>None</td><td>None</td></tr>
<tr><td></td><td></td><td></td><td></td></tr>
</table>

Overview	Properties	Security	Descriptors
Name: System ID **Description:** This characteristic represents a structure containing an Organizationally Unique Identifier (OUI) followed by a manufacturer-defined identifier and is unique for each individual instance of the product. **Type:** org.bluetooth.characteristic.system_id **Requirement:** Optional	**Property** / **Requirement** Read: Mandatory Write: Excluded WriteWithoutResponse: Excluded SignedWrite: Excluded Notify: Excluded Indicate: Excluded WritableAuxiliaries: Excluded Broadcast: Excluded ExtendedProperties:	None	None
Name: IEEE 11073-20601 Regulatory Certification Data List **Description:** This characteristic represents regulatory and certification information for the product in a list defined in IEEE 11073-20601. **Type:** org.bluetooth.characteristic.ieee_11073-20601_regulatory_certification_data_list **Requirement:** Optional	**Property** / **Requirement** Read: Mandatory Write: Excluded WriteWithoutResponse: Excluded SignedWrite: Excluded Notify: Excluded Indicate: Excluded WritableAuxiliaries: Excluded Broadcast: Excluded ExtendedProperties:	None	None

<table>
<tr><th>Overview</th><th>Properties</th><th>Security</th><th>Descriptors</th></tr>
<tr>
<td>Name:
PnP ID
Description:
The PnP_ID characteristic is a set of values used to create a device ID value that is unique for this device.
Type:
org.bluetooth.characteristic.pnp_id
Requirement:
Optional</td>
<td>
<table>
<tr><th>Property</th><th>Requirement</th></tr>
<tr><td>Read</td><td>Mandatory</td></tr>
<tr><td>Write</td><td>Excluded</td></tr>
<tr><td>WriteWithoutResponse</td><td>Excluded</td></tr>
<tr><td>SignedWrite</td><td>Excluded</td></tr>
<tr><td>Notify</td><td>Excluded</td></tr>
<tr><td>Indicate</td><td>Excluded</td></tr>
<tr><td>WritableAuxiliaries</td><td>Excluded</td></tr>
<tr><td>Broadcast</td><td>Excluded</td></tr>
<tr><td>ExtendedProperties</td><td></td></tr>
</table>
</td>
<td>None</td>
<td>None</td>
</tr>
</table>

Appendix C – Tx Power Service TBD

Name: Tx Power

Type: org.bluetooth.service.tx_powerDownload / View
Assigned Number: 0x1804

Abstract:
This service exposes a device's current transmit power level when in a connection.

Summary:
The Tx Power service is instantiated as a Primary Service. There is only one instance of the Tx Power service on a device. There is exactly one instance of the Tx Power Level characteristic

Service Dependencies
This service has no dependencies on other GATT-based services.

Transport Dependencies

Transport	Supported
Classic	false
Low Energy	true
High Speed	

Error Codes
This service does not define any application error codes that are used in Attribute Protocol.

Service Characteristics

<table>
<tr><th>Overview</th><th>Properties</th><th>Security</th><th>Descriptors</th></tr>
<tr>
<td>Name:
Tx Power Level
Description:
The Tx Power Level characteristic represents the current transmit power level of a physical layer for which the characteristic is associated.
Type:
org.bluetooth.characteristic.tx_power_level
Requirement:
Mandatory</td>
<td>
<table>
<tr><th>Property</th><th>Requirement</th></tr>
<tr><td>Read</td><td>Mandatory</td></tr>
<tr><td>Write</td><td>Excluded</td></tr>
<tr><td>WriteWithoutResponse</td><td>Excluded</td></tr>
<tr><td>SignedWrite</td><td>Excluded</td></tr>
<tr><td>Notify</td><td>Excluded</td></tr>
<tr><td>Indicate</td><td>Excluded</td></tr>
<tr><td>WritableAuxiliaries</td><td>Excluded</td></tr>
<tr><td>Broadcast</td><td>Excluded</td></tr>
<tr><td>ExtendedProperties</td><td></td></tr>
</table>
</td>
<td>None</td>
<td>None</td>
</tr>
</table>

Appendix D – Lock Command Control Service

Name: Lock Command Control

Type: Proprietary, Non-standard
Assigned Number: 0x1523

Abstract:
This service exposes the state and allows the control of the Lock when in a connection.

Summary:
The Lock Command Control service is instantiated as a Primary Service. There is only one instance of the Lock Command Control service on a device. There is exactly one instance of the Lock State and Lock Control characteristic

GATT Requirements

Sub-Procedure	Server Requirement
Read Characteristic Descriptors	Mandatory
Notifications	C1: Mandatory if the Lock State characteristic properties supports notification, otherwise excluded.
Write Characteristic Descriptors	C1: Mandatory if the Lock State characteristic properties supports notification, otherwise excluded.

Transport Dependencies

Transport	Supported
Classic	false
Low Energy	true
High Speed	

Service Characteristics

Overview	Properties	Security	Descriptors
Name: Lock State **Description:** The Lock Command Control characteristic is read using the GATT Read Characteristic Value sub-procedure and returns the current Lock state as a binary value; 0x00 represents an open lock, 0x08 represents an open lock with child safety removed. 0x01 represents a closed lock, 0x09 represents a weak closed lock state. **Type:** characteristic.lock_state **Requirement:** Mandatory	**Property** / **Requirement** Read — Mandatory Write — Excluded WriteWithoutResponse — Excluded SignedWrite — Excluded Notify — Optional Indicate — Excluded WritableAuxiliaries — Excluded Broadcast — Excluded ExtendedProperties	None	**Overview** / **Permissions** **Name:** Characteristic Presentation Format **Type:** descriptor.gatt.characteristic_presentation_format **Requirement:** if_multiple_service_instances **Permission** / **Requirement**: Read — Mandatory; Write — Excluded **Name:** Client Characteristic Configuration **Type:** descriptor.gatt.client_characteristic_configuration **Requirement:** if_notify_or_indicate_supported **Permission** / **Requirement**: Read — Mandatory; Write — Mandatory

Overview	Properties	Security	Descriptors
Name: Lock Command Control **Description:** The Lock Command Control characteristic is read using the GATT Read Characteristic Value sub-procedure and returns the current Lock state as a binary value; 0x00 represents an open lock, 0x08 represents an open lock with child safety removed. 0x01 represents a closed lock, 0x09 represents a weak closed lock state. **Type:** characteristic.lock_command **Requirement:** Mandatory	**Property** / **Requirement** Read — Mandatory Write — Mandatory WriteWithoutResponse — Excluded SignedWrite — Excluded Notify — Optional Indicate — Excluded WritableAuxiliaries — Excluded Broadcast — Excluded ExtendedProperties	None	**Overview** / **Permissions** **Name:** Characteristic Presentation Format **Type:** descriptor.gatt.characteristic_presentation_format **Requirement:** if_multiple_service_instances **Permission** / **Requirement**: Read — Mandatory; Write — Mandatory **Name:** Client Characteristic Configuration **Type:** descriptor.gatt.client_characteristic_configuration **Requirement:** if_notify_or_indicate_supported **Permission** / **Requirement**: Read — Mandatory; Write — Mandatory

Appendix D.1 – Lock State Attribute

Name: Lock State Bit Field
Type: Proprietary
Assigned Number: 0x1524

Abstract:
Categories of alerts/messages.

Summary:
The value 0x00 is interpreted as “Lock is open”.
The value 0x08 is interpreted as “Lock has been opened and child safety is removed, waiting for user to close. 15 seconds timeout applies”.
The value of 0x01 is interpreted as “Lock has been closed by user, lock in closed and locked state”
The value of 0x09 is interpreted as “Lock has been closed by user but lock is in error, lock will try to remove the error automatically”

Value Fields

<table>
<tr><th>Names</th><th>Field Requirement</th><th>Format</th><th>Minimum Value</th><th>Maximum Value</th><th>Additional Information</th></tr>
<tr><td>Lock State Bit Field</td><td>Mandatory</td><td>uint8</td><td>N/A</td><td>N/A</td><td>
Bit Field
<table>
<tr><th>Bit</th><th>Size</th><th>Name</th><th>Definition
<table><tr><th>Key</th><th>Value</th></tr></table></th></tr>
<tr><td>0</td><td>1</td><td>Lock State</td><td><table><tr><td>0</td><td>Open</td></tr><tr><td>1</td><td>Closed</td></tr></table></td></tr>
<tr><td>3</td><td>1</td><td>Alerts</td><td><table><tr><td>0</td><td>Normal</td></tr><tr><td>1</td><td>Warning / Error</td></tr></table></td></tr>
</table>
</td></tr>
</table>

Appendix D.2 – Lock Command Attribute

Name: Lock Command Bit Field
Type: Proprietary
Assigned Number: 0x1525

Abstract:
Categories of alerts/messages.

Summary:
The value 0x00 is interpreted as “Force the Lock to open”.
The value 0x01 is interpreted as “Force the Lock to remove the child safety mode, the user should now close the lock manually. 15 seconds timeout applies before the lock returns to the open state and the user is unable to close the lock manually”.

Value Fields

<table>
<tr><th>Names</th><th>Field Requirement</th><th>Format</th><th>Minimum Value</th><th>Maximum Value</th><th>Additional Information</th></tr>
<tr><td>Lock Command Bit Field</td><td>Mandatory</td><td>uint8</td><td>N/A</td><td>N/A</td><td>
Bit Field
<table>
<tr><th>Bit</th><th>Size</th><th>Name</th><th>Definition
Key</th><th>Value</th></tr>
<tr><td rowspan="2">0</td><td rowspan="2">1</td><td rowspan="2">Lock Command</td><td>0</td><td>Opening</td></tr>
<tr><td>1</td><td>Closing</td></tr>
</table>
</td></tr>
</table>

Appendix E – Device Firmware Update Service

TBD.

Appendix F – Test scenarios

Test Case Title	Current Status	Description	Expected Status	Muralis Comments	Test Completed
	Unlocked/Locked		Unlocked/Locked		Failed / Passed
Power-Up Sequence	Unlocked	Don't give any command and wait for 20 seconds.	Unlocked	The Lock shall remain idle.	
Uncompleted Locking Sequence	Unlocked	Give a Locking command and don't move the leaver, wait for 20 seconds timeout.	Unlocked	The Lock shall return to unlock secured after timeout. Making it impossible to move the leaver again	
Interrupted Locking Sequence	Unlocked	Give a Locking command, move the leaver halfway and wait for 20 seconds.	Unlocked	The Lock shall return to unlock secured after timeout. Making it impossible to move the leaver again	
Completed Locking Sequence	Unlocked	Give a Locking command and move the leaver to Locked position.	Locked	The Lock shall Lock successfully.	
Uncompleted Unlocking Sequence	Locked	Give an Unlocking command, block the leaver from opening into the Unlocked secured position.	Locked	The Lock shall put itself into the Locked condition again. Status always remained Locked during the test.	
Blocked Unlocking Sequence	Locked	Give an Unlocking command, block the leaver halfway from opening completely and move the leaver after sometime into the Locked position again.	Locked	Status will be "Unlocked" when lock is halfway and updated again when Lock is put into Locked position.	
Completed Unlocking Sequence	Locked	Give an Unlocking command.	Unlocked	Spring inside the Lock shall pull the leaver into the Unlocked secured position.	

Appendix G – Installing apps on Android

Installing apps on Android is relatively straightforward with the app store for Android, known as Google Play. You search for an app, select it and click install. These Android Apps are all signed and check for viruses and other harmful software before they are uploaded into the Google Play store. However, the Axa eRL Bluetooth Demo App is not available in the Android app store, not because it's not checked and/or signed but because it's a demo App. In this case you will have to manually download the App and install an .apk file. An .apk file behaves in a similar manner to an ".exe" file on Windows, you need to copy it to your devices Download directory and run it by clicking on it from your phone. Before you do this you have to enable installing apps from "Unknown Sources" explained in the following paragraphs.

Android 4.X

Here are some ways that you can manually install an application on Android without going through the app store. However, this is disabled by default for security reasons. Enable "Unknown Sources" Fig 1. Before attempting a manual installation of apps using the .apk files, you must first allow your phone to install from "Unknown sources" (i.e. non-app-store apps). To do this on Android 4.3 and 4.4, navigate to Menu -> System settings -> General -> Security and check the box marked "Unknown sources". In Dutch this is Menu-> Systeeminstellingen -> Algemeen -> Beveiliging and check the box marked "Onbekende bronnen".

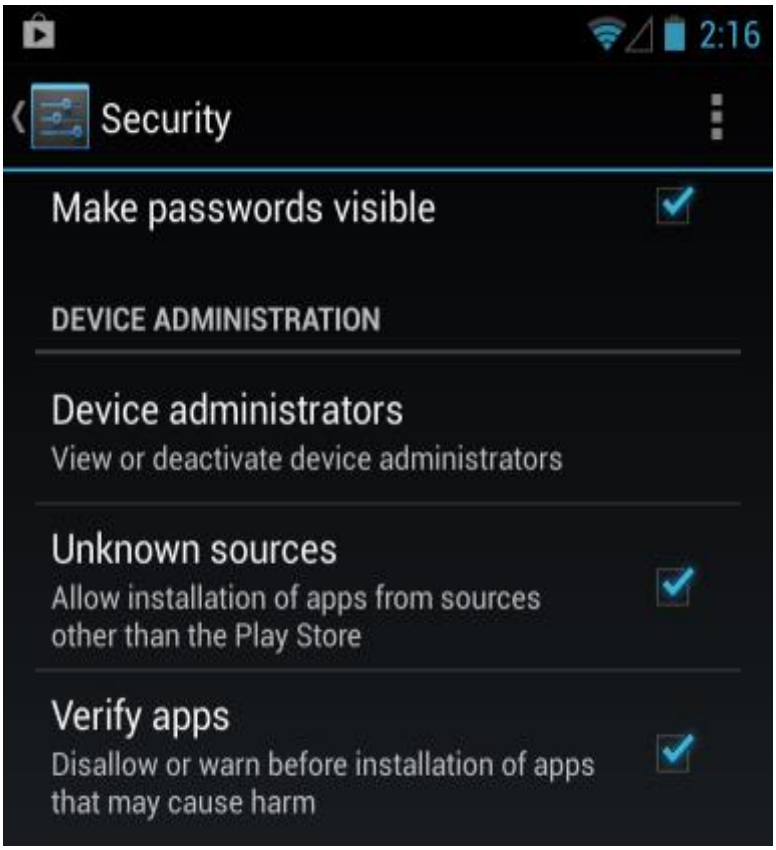

Figure 1.

Android 5.0

Installing the Demo App on Android 5.0 Lollipop is going a little bit different and easier.

- Go to your Android device's "**Settings**" and then **"Security Settings"**
- Search for an option "**Unknown Sources"**
- Check the box. A warning will appear, accept and allow the change
- You're all done!

Manually installing an application on Android without going through the app store, is disabled by default for security reasons. Enable "Unknown Sources" Fig 2. Before attempting a manual installation of apps using the .apk files, you must first allow your phone to install from "Unknown sources" (i.e. non-app-store apps). To do this on Android 5 Lollipop use the steps described above and mark the box "Unknown sources".

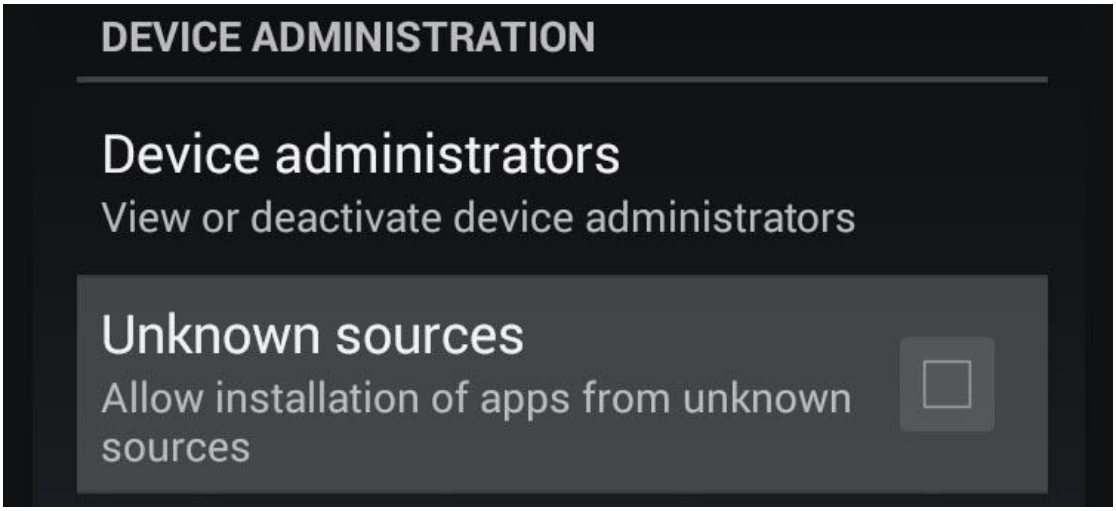

Figure 2.

Installation phase

When you have enabled installation from "Unknown Sources" you can proceed to installing de Axa Demo .apk file. Connect your phone to a computer via the USB interface and enable it to work like a memory stick. Now copy the .apk file to the Download directory on your phone, once it is copied and saved you should browse on your phone to the Download directory and click on the Axa Demo .apk file. Agree with the messages you get from the phone and the Axa Demo app will be installed. When installed you can delete the .apk file in the download directory since it's no longer being used.