SIC Project
Bluetooth-based, secure ad-hoc network for IoT devices

version 1.0

Authors: André Zúquete, Vitor Cunha

Version log:

- 1.0: Initial version

1 Introduction

Bluetooth is a wireless technology originally designed and first introduced in the late 90s that is meant to replace cables. The very first version was created to replace RS-232 serial cables, and that legacy still lives within Bluetooth. After many iterations and evolutions, Bluetooth is used today to connect a wide range of personal devices (e.g., earbuds, smartbands/smartwatches, home sensors, smartplugs, smartlamps, etc). In this project we will useit , namely its BLE version, it to implement a secure, ad-hoc network for IOT.

1.1 IoT (Internet of Things)

The term IoT often refers to a network where everything is connected to a single, universal network, the Internet. This allows devices anywhere in the world to communicate with services, or to receive order from services. This creates a very powerful way to manage and control things remotely, but also creates a set of security issues, namely in terms of access control.

In this project we will focus on creating a private IPv4 network of IoT devices that interact exclusively with a Sink host. The Sink is, for all practical purposes, a sort of applicational gateway for the applications running in the IoT devices.

The IoT devices can use only Bluetooth for their link layer. Cabled networks and other wireless communication technologies, such as Wi-Fi, are not available to them.

1.2 Ad-hoc networks

The term ad-hoc network is commonly used to refer to a network that gets formed and disappears without specif planning. They are formed by a set of (authorised) devices that happen to fulfill a given circumstance, such as being nearby. These networks are not common, though, they are mostly academic.

Ad-hoc networks typically make use of wireless communication technologies, since they make most sense to IoT devices that move. In theory, they can use any wireless technology they have access to; in this project, we will use only Bluetooth.

2 IoT ad-hoc network

Our IoT devices are both sensors (they gather information, which is provided to the Sink) and routers (they route traffic from a device to the Sink and vice versa). The Sink can have a direct communication with IoT devices on its reach, and an indirect communication with IoT devices beyond its reach. In this last case, there must exist a multi-hop communication through the network of IoT devices which implements a network layer.

3 Topology

The topology of the IoT network should be a tree, as shown in the figure below. Each IoT is n hops away from the Sink, with n ranging from 0 (direct connection) to any higher value. IoT devices should minimise the number of hops until the Sink, but following a lazy approach (instead of a greedy, or aggressive one). This means that when a device needs to enter a network to connect to the Sink, it must choose one nearby device (uplink) that presents the lowest possible hop count until the Sink (you can have many alternatives with the same hop count). But thereafter, they should not change the uplink while it works as such. Upon a loss of the uplink, the process of looking for a new one must be restarted.

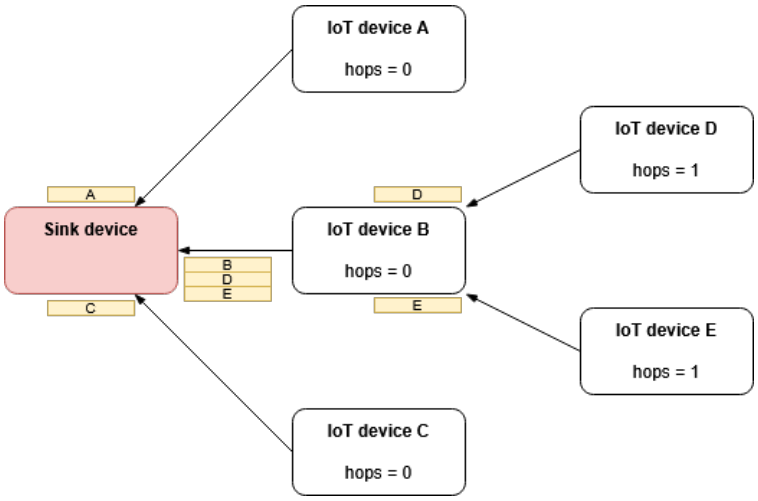

Figure 1: IoT ad-hoc network. The arrows represent bidirectional Bluetooth connections and their uplink direction. In yellow we have the network forwarding tables associated with Bluetooth connections.

When an IoT device looses its uplink, it should immediately break its connection to all the IoT devices that use it as uplink. This produces a chain reaction, effectively disconnecting all the devices that belong to a sub-tree that looses its uplink. For instance, if in the figure the IoT device B looses its uplink, it must consequently break the link with IoT devices D and E. When this happens, the disconnected devices should reenter the network as described above.

IoT devices without a current uplink should exhibit a negative hop count. This signals that no other IoT devices should use it as uplink, as it cannot work as such.

3.1 Addressing and routing

Each IoT device will use for its unique network identification a 128-bit identifier created during the device provisioning (see below). This identifier is also used for addressing. We will call it a NID (Network IDentifier).

The addressing should follow the fundamental approach of switches, and a similar exploitation of their forwarding tables. Each device memorizes the Bluetooth connection from which a message originally from a given NID arrived (in the uplink direction), and that connection will be use to send traffic in the opposite direction to the device with that NID. Thus, if E sends a message to the Sink, the message goes in the uplink to B, which memorizes the connection from which E's message arrived. Then, B sends the message in the uplink to the Sink, which memorizes the connection from which E's message arrived. When the Sink sends a message to E, it finds E in its forwarding tables and finds out that it should send it to B. B does the same, and sends the message to E, the target device.

3.2 Network liveness

A network Sink implements a heartbeat service. This service broadcasts (using multi-unicast) a hearbeat message at a given pace, such as once each 5 seconds. Each message contains successive counter value, and must be signed by the Sink.

These messages are not target at any device, they flood the network through the links from the Sink downwards, and then recursively downwards through the links to other IoT devices. Since the network is loop-free, these messages do not risk to be endlessly retransmitted. All IoT devices must verify the authentiticy of received heartbeats prior to used them locally and forwarding them through all downlinks.

Upon loosing 3 heartbeat messages in a row, an IoT device must consider its uplink to be down, must disconnect it, and also disconnect all the connections to other IoT devices it serves. Upon that, it must start looking for an alternative uplink to the Sink.

4 Network controls

Although the goal could be to maximize dynamism, you will implement a set of network controls in your IoT devices and Sink that help to debug and show the functionalities of your system. The controls are the following:

- Search nearby IoT devices and show their hop count until the Sink.
- Connect to a nearby IoT device. This should happen automatically, but it would be difficult to make different experiments with all devices very close to each other.
- Stop sending heartbeat messages to a given IoT device directly connected. This will be used to simulate a broken link between to IoT devices.

5 Security

5.1 IoT device's identification

Each IoT device is authenticated with an X.509 certificate. This certificate binds the NID to a public key. For public cryptosystem we will use a single elliptic curve (e.g. P-521). Elliptic curves are very convenient because they are fast, have a small memory footprint, and can be used in Diffie-Hellman key agreements, public key encryptions and digital signatures.

5.2 Sink identification

The Sink has a public key certificate just as any other device, issued by the same CA, but with a tweak: it contains a field in the Subject that identifies it with that attribute, besides having a NID.

5.3 Management of public key certificates

All IoT certificates must be produced by a central Certification Authority (CA), that should be part of the system. Consider, for instance, just for adding some real-world meaning to the system, that this CA belong to a company that is the owner of a large number of these IoT devices that are used for some business-oriented purpose (e.g., agricultural monitoring).

5.4 Bluetooth security

Bluetooth security covers only the protection (confidentiality and integrity control) of a direct link and the access control to the services provided by a device.

The link protection is determined by the pairing mode. We will use the simplest one, Just Works, which does not require mutual authentication, since this will be provided on a higher level with the certificates.

Regarding access control, the services of an IoT device are only available to other devices that have a valid certificate issued by a common CA. This requires a mutual authentication protocol among IoT devices after their connection using their public key certificates.

5.5 IoT device authentication and key distribution

The mutual authentication of IoT devices must produce a session key between them. Each authentication session must produce a different session key, even if one on the participants misbehaves.

5.6 Session key use

Session keys must be used to validate incoming messages as legit. For this purpose, their confidentiality is not critical (it is assured already by Bluetooth at the link layer) but their integrity is. Furthermore, message replaying can not occur, so senders and receivers must implement a strategy to detect and discard repeated messages. Appropriate MACs must be used in messages to assure their integrity and freshness (no replay).

5.7 End-to-end services

IoT devices provide data gathering for services. These services are identified by a name (instead of a number, as transport ports) and are provided by the Sink. In this project we will implement one single service, Inbox, to which arbitrary messages from IoT devices can be send to the Sink. Service clients are identified by a random number (similar to a transport port), and are use for downlink client addressing.

The end-to-end communication between each IoT device and the Sink must be protected with DTLS. DTLS is suitable to be explored over connectionless transport protocols, such as the one we are implementing in this end-to-end communication. For end-to-end authentication you should use the same certificates that are used for the authentication of peers in the direct links.

Each IoT device should have an internal service (say, a router daemon) that provides the basic networking features for the bidirectional communication with the Sink. This service listens all local Bluetooth connections (one uplink and one or more downlink) and forwards them if needed. The forwarding includes adding and removing per-link MACs, but not dealing with DTLS.

This service also deals with communications initiated by local clients with the Sink. For this purpose, the service may itself deal with the DTLS protection, which include secure channel agreements (with its counterpart in the Sink) and secure wrapping/unwrapping of traffic (generated locally or generated by the Sink, respectively).

In this project we will assume that there is only one Sink, instead of many. Therefore, the Sink identity and credentials do not change over time.

6 User interface

Each IoT device must have a user interface that shows the following information:

- The device NID;
- The uplink status, and the NID of the uplink device when connected;
- The list of downlinks and the NID of the downlink devices;
- The forwarding tables;
- The number of lost hearbeat messages;
- The number of messages routed through the uplink since it was established;
- The network controls (a sort of menu).

Each IoT device must also have the possibility to send a message to the Sink and this must show all the messages received, associated with the NID of the sender.

7 Implementation strategy

Given the features required for this project, it is advised that you implement it gradually, layer by layer, in order to test the underlying functionalities prior to develop others on top of them. Therefore, this a possible schedule for implementing your system:

- Develop the fundamental mechanisms to create and destroy Bluetooth connections between IoT devices;
- Develop the network controls;
- Develop the mechanisms to create public key certificates to IoT devices and to use them to negotiate session keys for Bluetooth links;
- Develop a basic message routing mechanism for sending data from an IoT device to the Sink, using a header with NIDs for device addressing, and names/numbers to identify services/clients and a MAC to check the authenticity and freshness of messages received by the devices;
- Develop the downlink broadcast mechanism that will help to implement the heartbeat protocol.
- Implement the timeout mechanism in the heartbeat to detect link failures.
- Implement the Inbox service, both in the Sink and in the IoT devices.
- Add DTLS to the message routing service.

We recommend using Linux hosts and the simpleBLE for handling Bluetooth BLE. We also recommend disabling all Bluetooth management applications (e.g. blueman in Linux) in the used hosts as these may interfere with your Bluetooth management actions.

Note: In this project you **cannot use** the Bluetooth BLE mesh networking standard (Mesh Profile and Mesh Model). This standard is based on network flooding, something that we want to avoid.

8 Project execution, delivery and grading

This work is expected to be implemented by a group of 4 students (regardless of their practical classes) and MUST be delivered using eLearning. Delivery should consist of a ZIP file with at least 4 folders and a file:

- sync: contains the code that is used exclusively in the Sink;
- node: contains the code that runs exclusively in the IoT devices that are not the Sink;
- common: contains the code that is common to IoT devices and the Sink;
- support: contains the code that handles support features, not exlored during the network operation, such as certificate issuing;
- README.md: contains a succinct report of the project design and implementation, which includes a justified description of all implemented features. It must clearly identify the authors (number and name), their percentual contribution to the overall project work and the features not implemented at all or partially implemented.

Projects will be graded according to:

- The suitability/correctness of the options taken for the generic management of the network (20%);
- The suitability/correctness of the security-related options and features (50%);
- The documentation produced (30%).

A maximum of 10% bonus wll be granted to projects that implement some interesting, non-mandatory features. As an example, you may deal with a scenario with multiple Sinks, which implies that each time you connect to a network you need to check if the Sink changed (in order to remove the DTLS secure channel to the Sink, that become unusable).

This project is expected to be authored by the students enrolled in the course. The use of existing code snippets, applications, or any other external functional element without proper acknowledgement is strictly forbidden. If any content lacking proper acknowledgment is found in other sources, the current rules regarding plagiarism will be followed.