Authentication and Authorization for Bluetooth LE Devices – Over The Air Download Firmware Update

Shreyas Gune | Fall 2015

Abstract

The age of Internet of Things is upon us. We find that all our devices are beginning to be labelled smart. This pertains to the fact that they have the ability to communicate some information over some network. This network is usually wired or wireless. This is the next step for the proliferation of connected devices and hence interconnected services that enable great new things for humans.

The applications are only limited by technology and human imagination. The technology is still in its nascent stages but promises to usher a new age of efficiency, convenience and activity in many different industries and residential settings. Bluetooth Low Energy is one of the prominent technologies by which this Internet of Things is implemented. It is not so different from Bluetooth classic in its layered architecture, but differs greatly in terms of power consumption and this fact enables a world of possibilities for small devices that either allow remote communication for the mobile devices or notify information to them. In light of the scheme of implementing such applications and services onto devices and the applications that communicate with the said devices, may it be heart-rate monitor, medical analysis of important tests or security door locks, among many other applications, there is a concern for security of access and threat of manipulation of values.

In my project, I have used Texas Instruments SensorTag device that houses a range of many different sensors that can be implemented to send out data from their target space. The device functionality also offers developers the ability to flash new firmware images that implement the said applications, which makes it easy to update any changes or upgrades, all through the convenience of your mobile phone.

The issue that is identified is of security, which enables any person with enough knowledge and set of resources to flash a custom image onto the device without any prior authentication measures to do so. Ideally, the Sensor Tag must verify that the mobile device requesting the ability to update the firmware is the authorized one. This is currently not available in the application service offered by iOS and Android implementation.

I have attempted to mitigate this situation by designing my custom algorithm, creating a firmware specific service on the SensorTag and a custom application on the Android OS to implement the said algorithm and explored the limitations offered by the TI RTOS, Bluetooth LE Stack and Android API's in the attempt to implement this algorithm, as well as surveyed the validity of TI's SensorTag in the market, but collecting power consumption statistics over a definite period.

INDEX

Chapter 1

Internet of Things

There is a new wave of physical objects, casually referred to as things that are proliferating the industry and market. These things are small scale devices that embedded elements that comprise of electronics, software, sensors and network connectivity. The network connectivity is leveraged on multiple interfaces to enable local and remote objects to collect and exchange data. So, largely, the inter-network of these physical objects capable of dynamic data collection and exchange is known as Internet of Things (IoT).

This architecture allows singular/multiple devices to sense objects and control them remotely across existing wireless network infrastructures. This ends up creating opportunities which enables an atypical interaction between physical and computer-based systems. These interactions are then implemented with modifications to deliver efficiency, accuracy and might even seem particularly beneficial from an economic and social standpoint. IoT offers advanced connectivity between devices, systems and services that supersedes machine-to-machine communications and stretches out to many protocols, domains and applications. This results in the automation of singular task-related events in daily life as well as an industry wide implementation. The amount of data flow rises with the rise in the number of devices and this also provides an avenue into the field of "Big Data".

The expert estimation predicts about 26 billion objects by the year 2020.

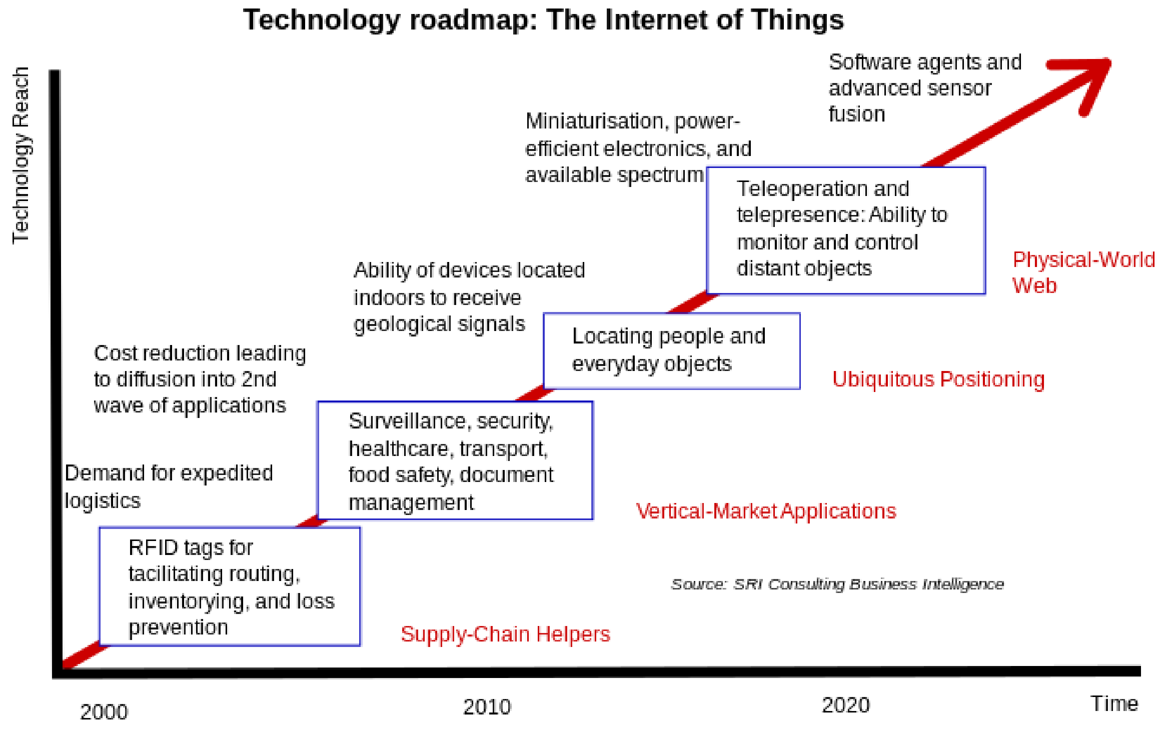

Technology roadmap: The Internet of Things
Technology Reach
Software agents and advanced sensor fusion
Miniaturisation, power-efficient electronics, and available spectrum
Teleoperation and telepresence: Ability to monitor and control distant objects
Physical-World Web
Ability of devices located indoors to receive geological signals
Locating people and everyday objects
Ubiquitous Positioning
Cost reduction leading to diffusion into 2nd wave of applications
Surveillance, security, healthcare, transport, food safety, document management
Vertical-Market Applications
Demand for expedited logistics
RFID tags for tacilitating routing, inventorying, and loss prevention
Source: SRI Consulting Business Intelligence
Supply-Chain Helpers
2000
2010
2020
Time

Figure 1 – Technology Roadmap (IoT)

In the future, it is projected that the IoT project will adopt a more non-deterministic and open network, which will primarily consist of auto-organized entities. These entities may include Web Services, Service Oriented Architecture components, virtual objects. The contexts defined for these objects will enable them to act independently and will respond to their stimulus based on circumstances. This constructs a premise for machine learning based autonomous behavior methods based on collection of data and reasoning context information.

Architecure

The system adopts a variation of event-driven architecture, from the bottom up. This will comprise of processes and operations performed in real-time, based on context and supports model driven approach as well as functional approach. The event needs no deterministic or syntactic model, but will vary from context to context. This requires huge scalability on the network architecture with space to handle the surge of devices. IETF 6LoWPAN would be used to connect devices to IP networks. With the large number of devices connected, they each will have unique addressing and IPv6 will meet the scalability requirements. Data transport will be handled by IETF's Constrained Application protocol, MQTT and ZeroMQ.

In terms of standardization and size consideration, the internet of objects would encode 50 to 100 trillion objects and be able to follow movement of those objects. Human beings in surveyed urban environments are each surrounded by 1000 to 5000 trackable objects.

Space considerations also play a critical role as each device will take up geographical space. Each device is now a stand-alone processing entity and it's location and time synchronization data is of key importance. This urges attention on time-space context to be given one of the central roles in the said information eco-system, and calls for standardized geospatial standards, akin to web standards.

Simulations for IoT networks can be performed using OPNET, NetSim and NS2.

Targets

The core targets of Internet of Things project are enterprise, home and government. By 2019, the Enterprise Internet of Things sector is estimated to account for nearly 40% or 9.1 billion devices.

Technologies

There are 3 major technologies that enable IoT :

1. RFID and NFC(Near Field Communication)- a set of protocols that enable electronic devices to establish radio communication with each other by touching devices together or bringing them in a close proximity (~10cm) or less.

2. Optical Tags and Quick Responses - QR Codes.

3. Bluetooth Low Energy – All new smartphones are comprised of BLE hardware. Tags based on BLE can communicate with applications built on smartphones and conduct various tasks. The low energy based data exchange will ensure operation for longer durations on modest battery usage.

Security Challenges

There are concerns reverberating across the community on the irregular considerations on security measures while promoting the Internet Of Things project. There are many devices that play a crucial role in the control and automation industry such as ones used on automobile industry, for car brakes, engine start-stop, locking systems etc. One such challenge has been identified in the authentication scenario, as the Bluetooth Smart (Bluetooth LE) does not have authentication service implemented in its 4.1 stack.

Chapter Two

Bluetooth Low Energy

Introduction

Bluetooth Low Energy (also known by it's market name : Bluetooth Smart) is essentially a WPAN (wireless personal area network technology) designed by Bluetooth Special Interest Group, aimed at leveraging and enabling the Internet of Things project that was defined in Chapter 1. In this report, Bluetooth LE and Bluetooth Smart mean the same thing and will be used interchangeably.

It is an emerging technology, and has not yet offered any kind of support for backward compatibility for previous Bluetooth versions, although the Bluetooth 4.0 specification has permitted the implementation of either Bluetooth Classic or Bluetooth LE or both based on the context of the implementation.

Technical Details

Bluetooth Smart shares the same spectrum range as the one used by Bluetooth classic which is 2.400 GHz to 2.4835 Ghz ISM band. The only difference is that a different set of channels are used. While Bluetooth Classic uses 79 1-MHz channels, Bluetooth Smart uses 40 2-MHz channels. The modulation technique used is Gaussian Frequency Shift Keying (GFSK) supporting frequency hopping.

Data transfers only support very short data packets comprising of 9 octets in the minimum to 27 octets in the maximum transferred at 1 Mbps and connections use advanced sniff-sub rating to achieve ultra low duty cycles. Taking an example, if we have a radio technology that have packets that are 3 ms long, it takes considerable amount of energy to keep sending those packets at same frequency. As the transmission occurs, the actual silicon in the chip is heating up, that changes the characteristics of the silicon, which changes the frequency. While transmission, the radio needs to be recalibrated on multiple occasions and to avoid this, we have the coin-cell operation.

Achieving low-power itself is a mammoth task, which involves three things – lowering standby times, which involves keeping the radio off, so lower duty cycles; Faster connection involves less time the radio is on and thirdly lower peak power which essentially are provided by coin cell battery. These cell batteries are designed to never draw more than 20milli amp hours. That is maximum that Bluetooth Low Energy is allowed to, and hence designed to work on.

Technical Specification	Classic Bluetooth technology	Bluetooth Smart technology
Distance/Range (theoretical max.)	100 m (330 ft)	>100 m (>330 ft)
Over the air data rate	1–3 Mbit/s	1 Mbit/s
Application throughput	0.7–2.1 Mbit/s	0.27 Mbit/s
Active slaves	7	Not defined; implementation dependent
Security	56/128-bit and application layer user defined	128-bit AES with Counter Mode CBC-MAC and application layer user defined
Robustness	Adaptive fast frequency hopping, FEC, fast ACK	Adaptive frequency hopping, Lazy Acknowledgement, 24-bit CRC, 32-bit Message Integrity Check
Latency (from a non-connected state)	Typically 100 ms	6 ms
Minimum total time to send data (det.battery life)	100 ms	3 ms [31]
Voice capable	Yes	No
Network topology	Scatternet	Star
Power consumption	1 W as the reference	0.01 to 0.5 W (depending on use case)
Peak current consumption	<30 mA	<15 mA
Service discovery	Yes	Yes
Profile concept	Yes	Yes
Primary use cases	Mobile phones, gaming, headsets, stereo audio streaming, smart homes, wearables, automotive, PCs, security, proximity, healthcare, sports & fitness, etc.	Mobile phones, gaming, PCs, watches, sports and fitness, healthcare, security & proximity, automotive, home electronics, automation, Industrial, etc.

Sniff sub-rating is a Bluetooth feature designed to increase battery life of devices as much as 500% for the ones that are not active for long durations. This rating reflects the inter-decision between devices on the duration of wait time before they send out keep-alive messages to each other. As opposed to previous exchanges occurring several times per second, as defined in the 2.1 specification, the inter-device negotiations only take place once every 5-10 seconds.

Bluetooth LE uses only 3 advertising channels as opposed the Bluetooth classic that takes about 16-32 channels. The 3-advertising channel mode just requires the radio to be on for 0.6 to 1.2 ms instead of the mammoth 22.5 ms found in Bluetooth classic so there is 20 times reduction in radio-on time.

Connections are enabled in 3 ms and peak power consumption has been reduced by increasing the modulation index on GFSK, and also offers better range and better robustness.

Architecture

Bluetooth LE is implemented using a Layered Architecture.

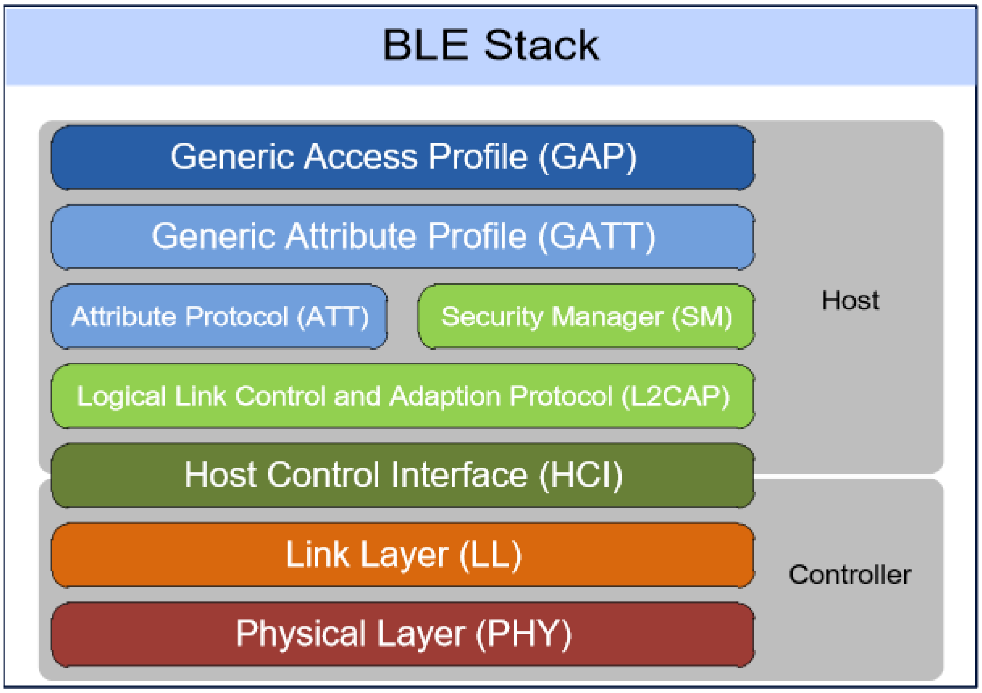

BLE Stack
Generic Access Profile (GAP)
Generic Attribute Profile (GATT)
Attribute Protocol (ATT)
Security Manager (SM)
Logical Link Control and Adaption Protocol (L2CAP)
Host Control Interface (HCI)
Link Layer (LL)
Physical Layer (PHY)
Host
Controller

Figure 2 – Bluetooth Low Energy Stack

Physical Layer – transmits and receives bits of information over a wireless medium.

Link Layer – receives this bits and sees them as packets and offers protocols to control these packets sending.

HCI – it's an Host/Controller interface allowing one to have 'swappable parts' , which means you can swap out the chips based on your whims and fancy without much hassle.

L2CAP – Above the controller, there are different channels and each of them performs a different purpose. L2CAP is a multiplexing layer between those channels. It manages 3 fixed channels. Two of them are dedicated for higher protocol layers like ATT, SMP. One is used for LE-L2CAP protocol signaling channel for custom use. It also supports connection oriented channel over an application that is specifically registered using protocol service multiplexer channel.

Attribute Protocol – Reading and Writing data on the device. Defines a client/server architecture above BLE logical transport channel. It allows a device to expose a set of attributes and associated values to peer devices. These can be discovered, read and written by a client.

Attribute Profile – Defines the services and meta-attributes on the device.

Security Manager – Defines the procedures and behavior to manage pairing and encryption between devices.

Device Profiles – What we could actually do on these devices (leveraging data registers, sensors etc)

Modes of Operations:

Dual Mode – have BR/EDR as well as LE.

Single Mode – only supports LE technology

Single Mode Bluetooth Basic Rate	**Single Mode Bluetooth Low Energy**
Serial Port Profile/OBEX/BNET	Attribute Profile
RFCOMM Protocols	Attribute Protocol
L2CAP	L2CAP
Link Manager	Link Layer
Basic Rate RF	Low Energy RF

For dual mode, L2CAP is shared. Android, iOS and Windows Phone devices are dual mode devices.

Dual Mode

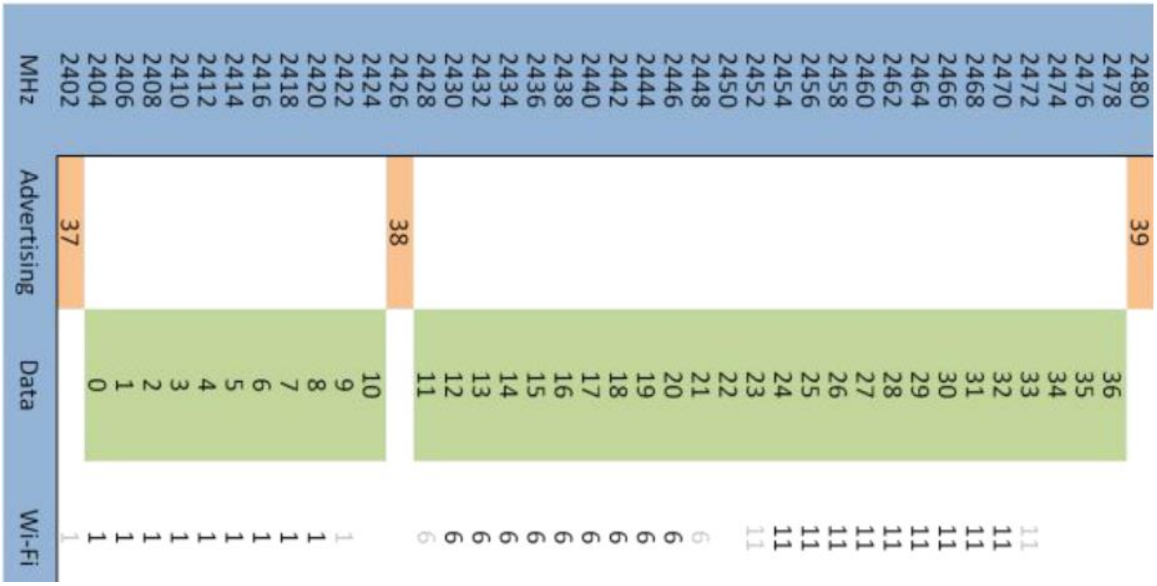
Serial Port Profile/OBEX/BNET	Attribute Profile
RFCOMM Protocols	Attribute Protocol
L2CAP	
Link Manager	Link Layer
Basic Rate RF	Low Energy RF

Physical Layer Specifications

Advertising Channels : 3 fixed channels

- Used for broadcast data
- Connectable advertisement
- Discoverable advertisement

Data Channels : 37 dynamic channels, adaptively frequency hopped



MHz	Advertising	Data	Wi-Fi
2402	37		1
2404		0	1
2406		1	1
2408		2	1
2410		3	1
2412		4	1
2414		5	1
2416		6	1
2418		7	1
2420		8	1
2422		9	1
2424		10	
2426	38		
2428		11	6
2430		12	6
2432		13	6
2434		14	6
2436		15	6
2438		16	6
2440		17	6
2442		18	6
2444		19	6
2446		20	6
2448		21	6
2450		22	
2452		23	11
2454		24	11
2456		25	11
2458		26	11
2460		27	11
2462		28	11
2464		29	11
2466		30	11
2468		31	11
2470		32	11
2472		33	11
2474		34	
2476		35	
2478		36	
2480	39		

Figure 3 – ISM Band Channels

Advertising Channels on 37, 38 and 39 , are chosen specifically to avoid WiFi (which by default choose channel 1, 6 and 11)

GFSK Modulation with BT product equal to 0.5 and Modulation Index 0.5, which is similar to GMSK.

Link Layer Specifications

Stand-By State : Idle

Scanning State : Looking for advertisement

Initiating State : Looking for connection to a particular device.

Advertising State : Looking for someone to connect to the generated advertisement.

Connecting State

Example –

A sends out advertisements. “Hi, I’m A, please connect to me”.

B is in initiating state. B is listening for advertisement. B sends a message out to A and enters a connecting state as a Master.

A hears this message and also enters the connecting state, but as a Slave.

Initiator → MASTER and Advertiser→SLAVE

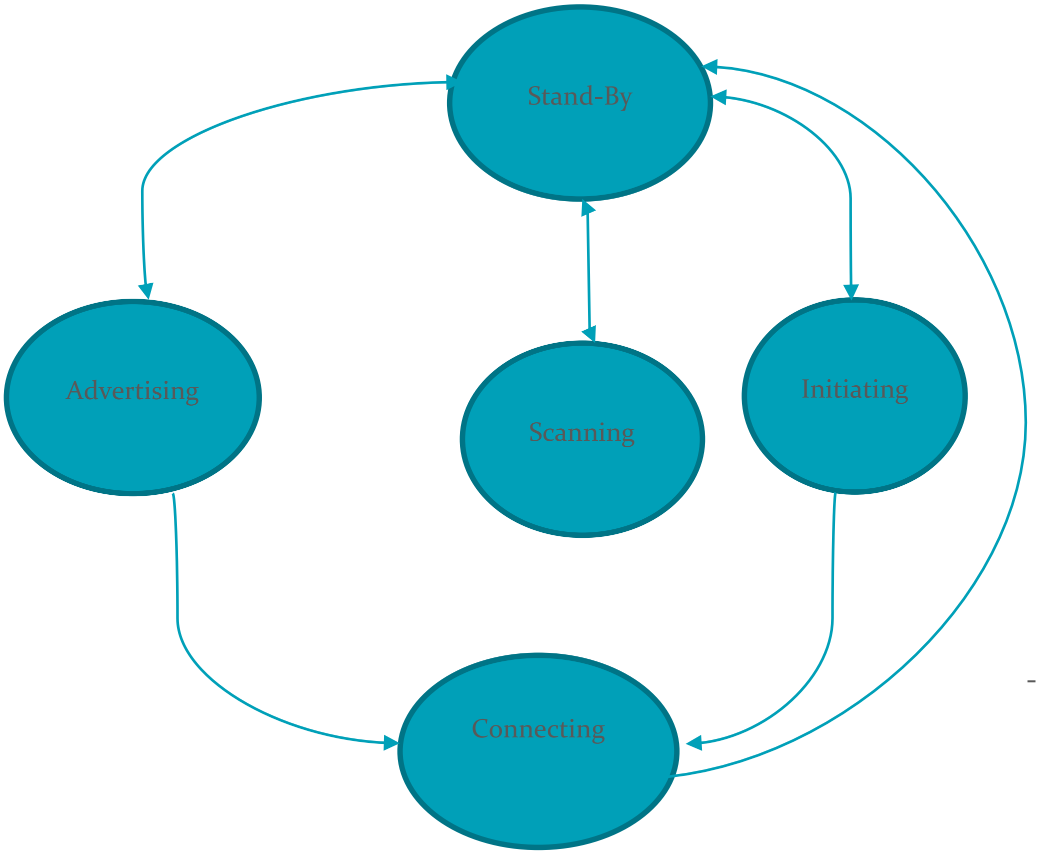


Figure 4- Link Layer State Diagram

Sequence Charts

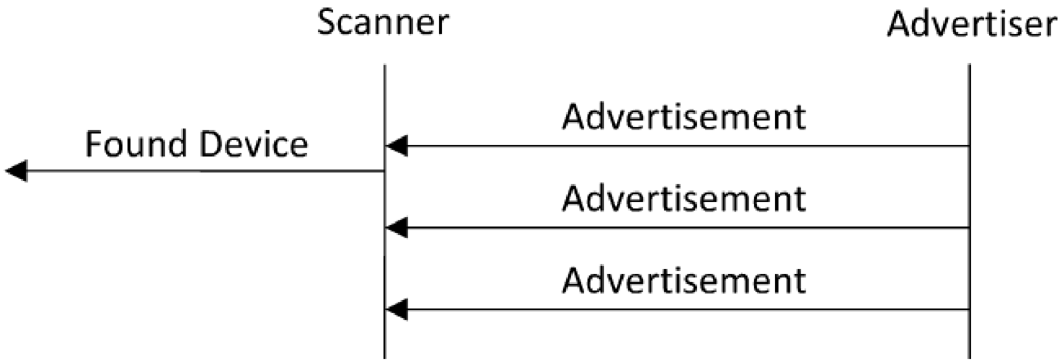


Figure 5 – Passive Scanning

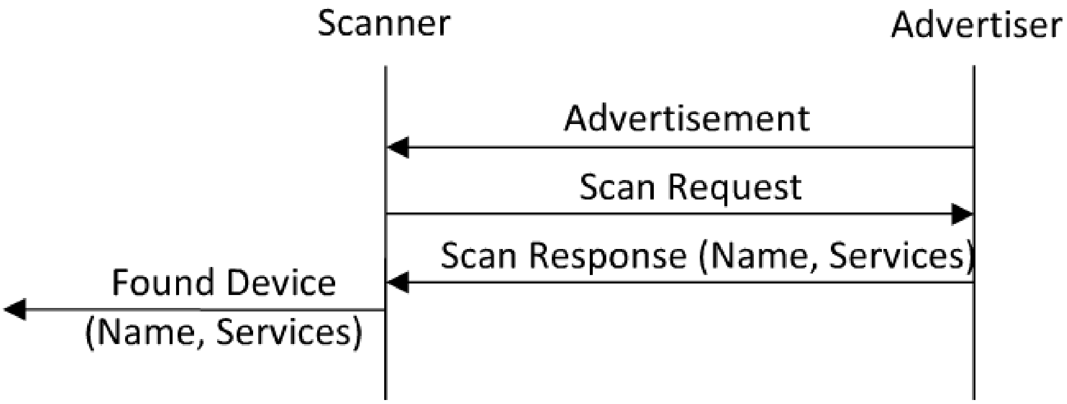


Figure 6 – Active Scanning

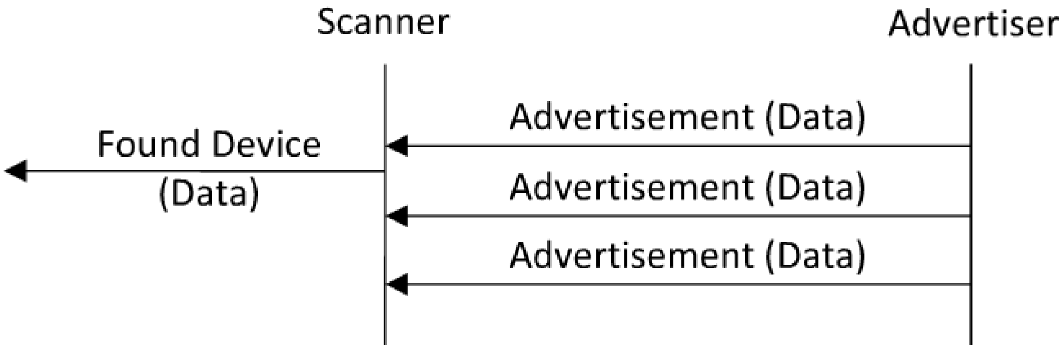


Figure 7 – Broadcasting Data (only)

Example : In a train-station, the data being broadcast about the train information to the device, where GPS signal is poor.

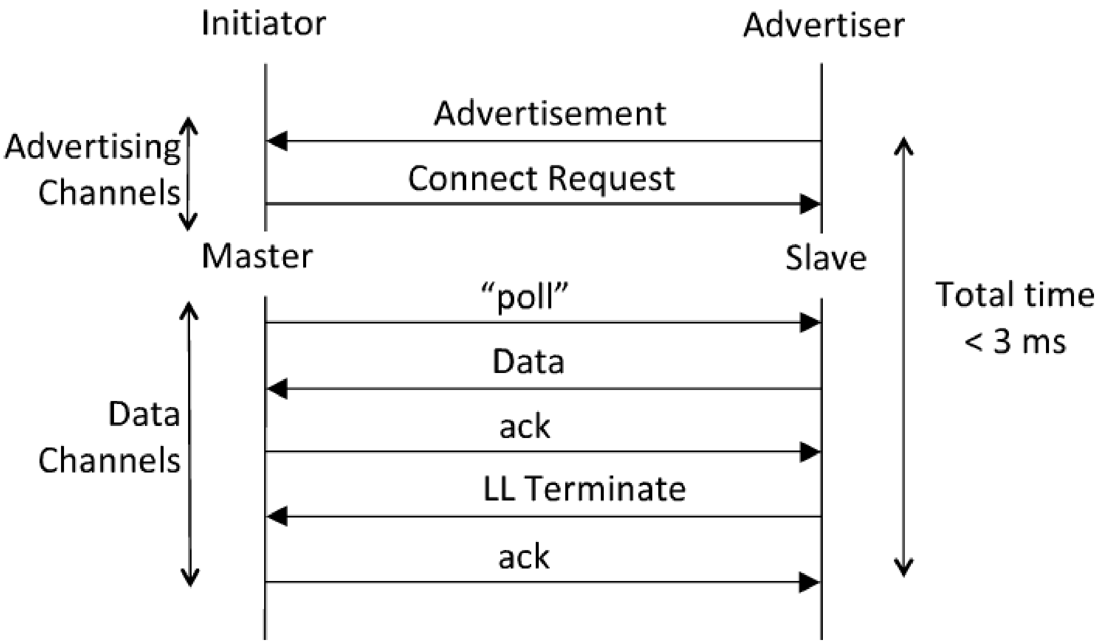


Figure 8 – Initiating Connection

A master can initiate a connection to a list of devices, whenever the advertisement is heard. When the imitator hears the advertisement, it sends a connection request.

Connection request includes information like instance of data transmission, sniff interval, how often is the initiator, not supposed to listen to the device, sub-rate, adaptive frequency-hopping map, instance of changing the frequency hop pattern, etcetera.

Actual time that the radio is on in this 3ms transaction period, is 1-1.5 ms. That is how low the power usage is.

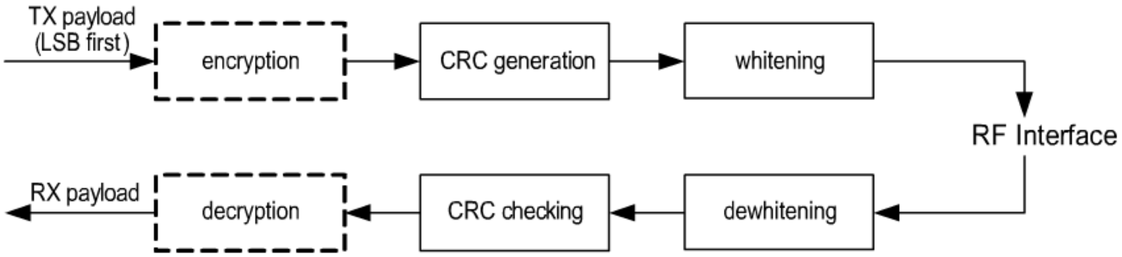
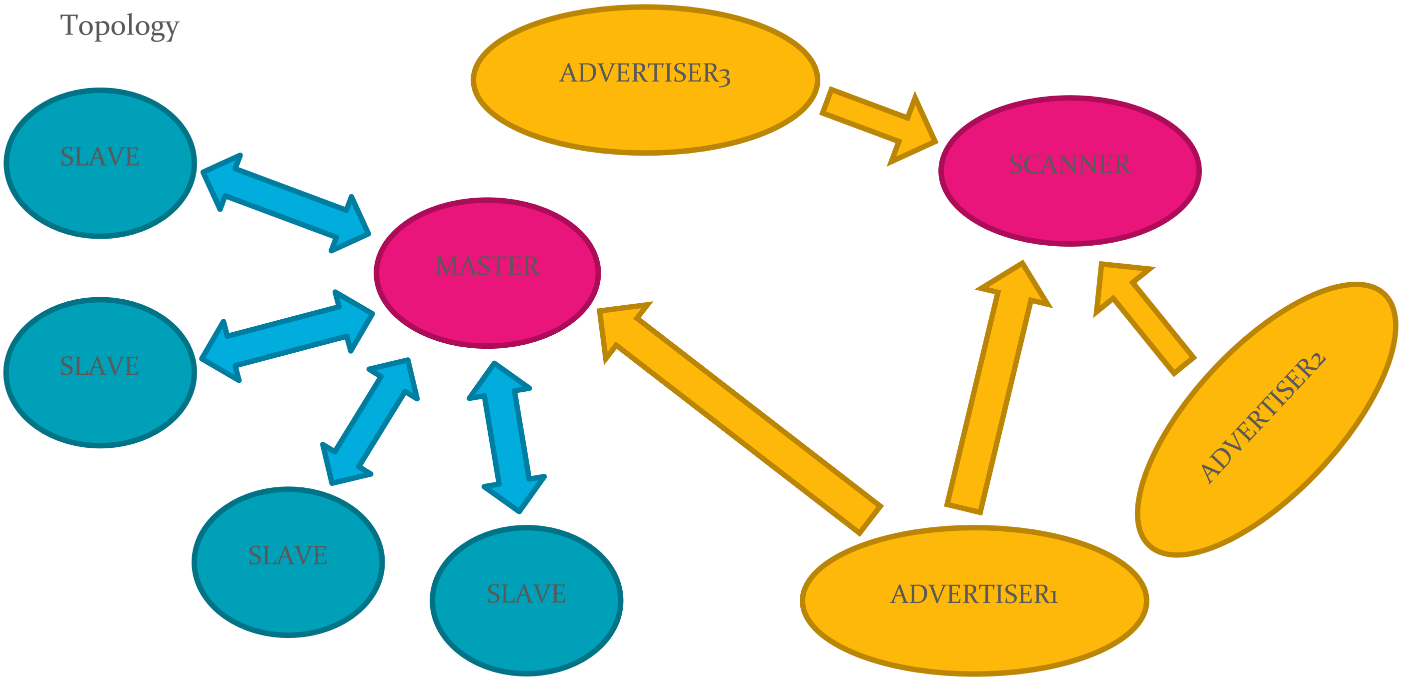


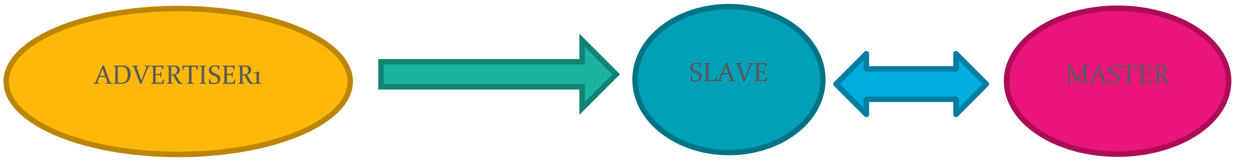
Figure 9 – Bit Streaming

The encryption and decryption is conducted before the CRC in order to reduce strain on the transmission time.

Topology



Advertiser 1 sends out an advertisement. MASTER hears it and sends out a connection request. And now, MASTER has ADVERTISER1 as a new SLAVE :



Packet Structure

Preamble : 01010101 or 10101010 | Access Address : 32 bits | Payload (2-39 octets) | CRC : 24 bits

Preamble does an automatic gain control to successfully receive the rest of the packet. 16 bit CRC is not good enough in noisy-environment, for example, an industrial power plant. 24 bit CRC provides a more robust protection.

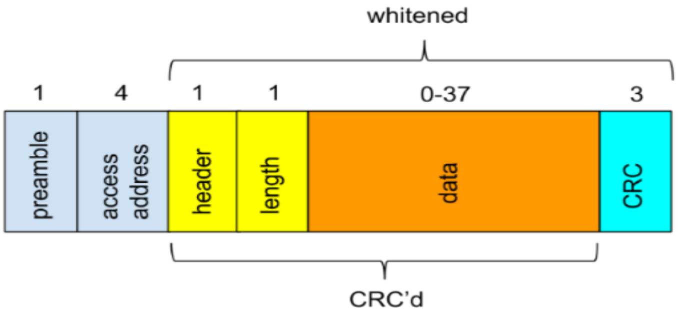


Figure 10 – Packet Structure

Security on Link Layer : The payload is encrypted using AES 128 scheme. Although, the security is unbalanced. According to Bluetooth specification, slave devices are very resource constrained, operating on cell battery and has very limited or low storage. Therefore all keys, identities and authentication information has to be handled by the master. The security issues and hacking Bluetooth Low Energy will be observed in detail in the next chapter.

Generic Access Profile

The GAP layer of the BLE protocol is mainly responsible for connection functionality. It takes care of access modes and procedures.

Connectable Modes – Device is able to make a connection. (connectable/non-connectable)

Discoverable Modes – Device can be discovered by other devices (advertiser) (non-Discoverable, limited discoverable, general-discoverable)

Bondable Modes – Pairing with a device and have a long term relationship. (non-bondable, bondable)

Name Discovery – Find the name of other devices. (name shared between dual mode device supporting BR/EDR . Advertising data contains name.

Device Discovery – Search for a particular or all devices. (Finds all addresses and name of devices)
Profile Roles : Central, Peripheral, Observer, Broadcaster.

Link Establishment – Selecting (connecting) to a device. It is connected by sending a CONNECT_REQ packet.

Service Discovery – After Link Establishment, discovering supported profiles.

Different Roles assumed by a device :

- Standby : Idle state is entered upon device reset.
- Advertiser : Advertises with data specific to its information, letter initiating devices know that this advertiser is ready to be connected to , ergo this advertiser is indeed a connectable device.
- Scanner : The scanning device receives the advertisement and sends a scan request. This is responded to by the advertiser with a scan response. This is known as device discovery and now, scanning device is aware of the fact that it has found a connectable device to initiate a connection.
- Initiator : The device specifies a peer device address to connect to. If an Advertisement is received matching that peer device's address, the initiating device will then send out a request to establish a connection link with the advertising device with connection parameters.
- Slave/Master : Once a connection is formed, the device will function as a slave (if it was an advertiser) or a master (if it was the initiator).

Connection Parameters include connection interval, where as we have understood, that two devices agree upon a channel, exchange data and then agree to be connected on a different channel after some amount of time and exchange some data. The hopping pattern is pre-decided and the meeting and exchange of data between two devices is known as connection event. In order to keep the connection established, the devices keep exchanging the link layer data, if there is no real application layer data to exchange. The time between the dying of one channel connection and the being alive of the other channel is known as connection interval. Its value can be 6-3200.

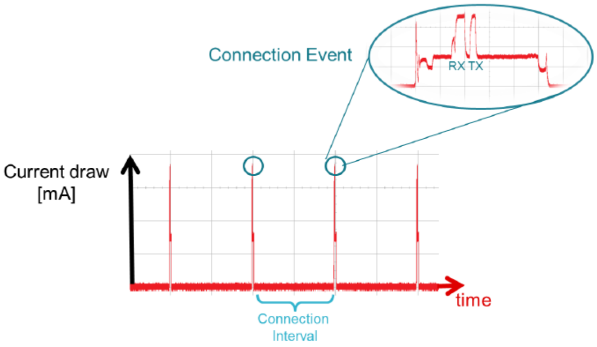
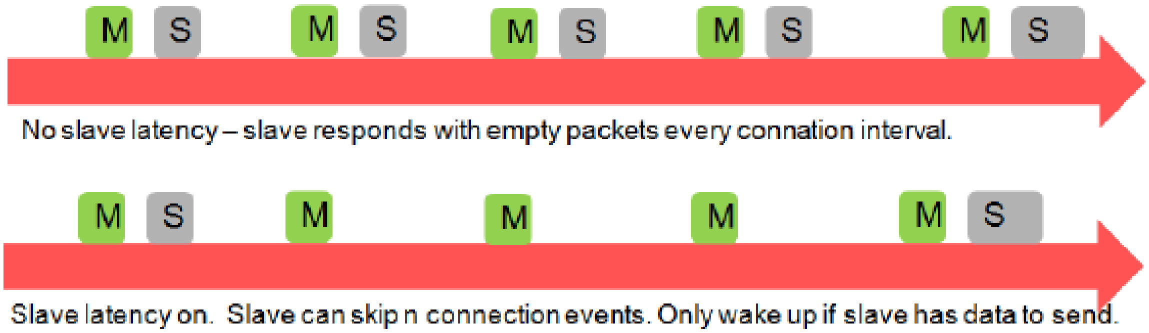


Figure 11 : Connection Interval

Slave Latency represents a scenario where a peripheral device can skip a fixed amount of connection events. This essentially saves the device the necessity to exchange link-layer data and strain of maintaining a connection. The skipping range is from 0 to 499.



No slave latency – slave responds with empty packets every connation interval.

Slave latency on. Slave can skip n connection events. Only wake up if slave has data to send.

Figure 12 : Slave Latency

In a scenario when the device connection is lost, there is still a counter that is monitoring time taken to establish a new successful connection. This is called supervision timeout.

In case a device assumes slave latency, the effective connection interval then becomes the product of connection interval and (1+slave latency).

For example :

```
CI = 50 (62.5 ms) //Connection Interval
```

```
SL = 4
```

Therefore effective `connection interval = 62.5*(1+4) = 312.5 ms`. So when no data is being sent from a slave to its master, the slave will only transmit during a connection, once every 312.5 ms.

Reduction in connection interval and slave latency causes increase in power consumption for both devices, throughput and time taken to transfer data, while increasing the connection interval as well as slave latency will cause a reduction in said parameters and increase the amount of time to transfer data in the case of slave latency increment.

Chapter Three

Bluetooth Security and Hacking Bluetooth LE

Security Manager Protocol

Key usage in Bluetooth LE is a stack level authentication process that involves an Identity Root which happens to be a 128-bit random number, which is proposed for security measure. The Identity Resolving Key (IRK) is a 128-bit value is derived from the Identity Root by

$$\mathtt{IRL} = E_{IR}(\text{‘1’})$$

And is used to generate and or resolve a random address. The Diversifier Hiding Key (DHK) is a 128-bit number that is hides a Diversifier which is a 16-bit number that is used for device authentication.

The Encryption Root is a 128-bit random number that is allocated by the link controller and is used by the device that is in the slave role. From the encryption root, the LTK (Long Term Key) is derived. It is a 128-bit value and generates a Short Term Key (STK) which is used to encrypt connection. The Long Term Key is stored in the master.

$$\mathtt{LTK} = E_{ER}(\mathtt{DIV})$$

Phase 1 : Request for pairing : Unencrypted , simply asks to pair with the other device. The IO capabilities are exchanged in this phase, authentication information is shared with bonding preferences which particularly refers to phase 3.

Phase 2 : The authentication algorithm is chosen based on request. This is also unencrypted.

Out of Band – if both devices support OOB IO capabilities, TK = OOB_data.

Pin Entry – if devices support : Keyboard and a display : TK = pin_value

No_Authentication – if device supports a display, TK = 0

A Short Term Key is derived.

Phase 3 : Keys are generated and the link is secured, as the slave. This is a special bonding-mechanism. Slave sends the Long Term Key, DIV and IRK.

Setting Up Security :

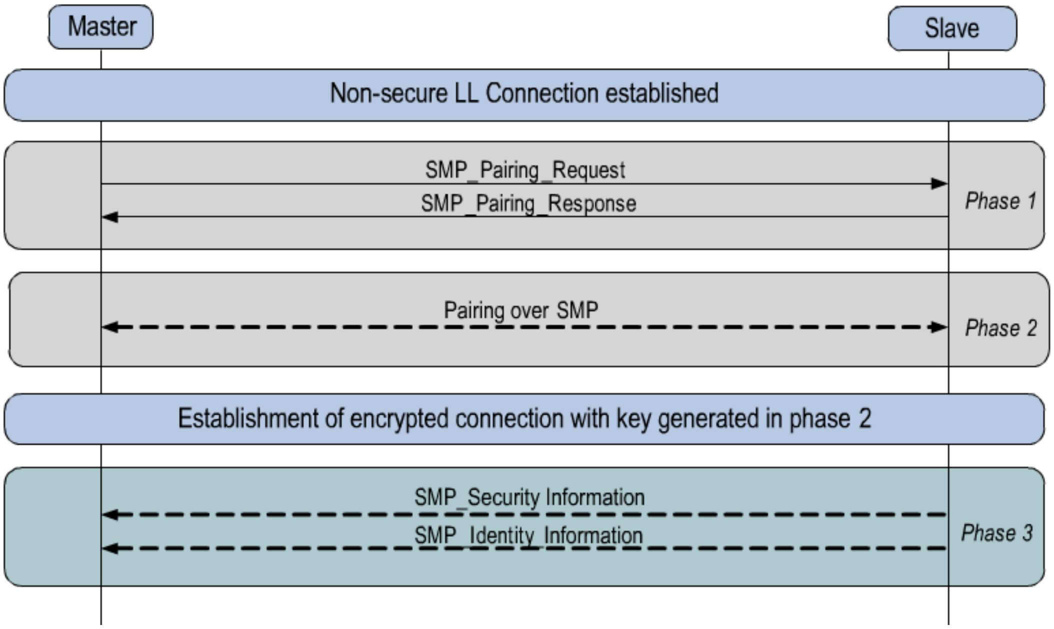


Figure 13 – Security Management Protocol

Hacking Bluetooth LE

Sniffing_Bluetooth classic is hard, but sniffing Bluetooth LE is slightly easy. As we have observed in the Bluetooth Low Energy stack, PHY→Link Layer→L2CAP→ATT→GATT.

PHY layer carries an RF on GFSK modulation with 40 x 1 MHz channels spaced 2 MHz apart.

Project Ubertooth is a Bluetooth packet sniffing open source hackerspace project that is a culmination of hardware, firmware and host code. The hardware design is available for anyone with an internet connection to download the schematics and build one at home or buy one that's available by a seller. The firmware is written for the ARM processor and assumes that the USB bootloader plus Bluetooth_rxtx firmware is already installed on the board. The host is connected to the sniffer dongle and running code that is optimized and customized to analyse the RF from the air-interface. Once the antenna is connected to dongle, the RST and 1V8 LEDs illuminate. This is an indicator of LPC175x microcontroller is actively running (RST) and that the CC2400 wireless transceiver IC (1V8) has power. All source code for the firmware and host analysis software is open source and easily configurable.

So the Ubertooth dongle takes a chunk of RF and gets RAW bits. The link-layer packet looks like

```
PREAMBLE(1 octet) | Access Address (4 octets) | PDU (2-39 octets) | CRC (3 octets)
```

As a sniffer, we desire the start of the PDU. That's how we access the data unit. So in the stream of bits that is sniffed, we look for the 32 bit value, which most likely is the start of the packet. From the start of the packet, we know where the start of the access address is and the octet boundary, when we go forward 32bits and we obtain the start of the PDU.

Here is a sample of a Bluetooth LE packet as obtained by Ubertooth.

```
06 0b 07 00 04 00 1b 11
00 16 58 b8 02 62 fb b2
```

which breaks down into :

```
06 0b
  07  00                  //L2CAP Length
  04  00                  //Channel 4: LE Attribute Protocol
     1b                   // Handle Value Notification
     11  00              //Attribute Handle
       16                //Flags
         58              //heart-rate sensor : 88 bpm
         b8  02          //RR-interval : 696 ms
62  fb  fb
```

So now, we can obtain RF into packets. Next step is to follow the connection.

As we know, there are 37 data channels and the master and slave hop along a fixed hopping pattern, which is a fixed pattern, where there is a fixed value added and the master-slave emit one packet per time-slot.

Example: 3→10→17→24→31→1→8→15… where the hop amount is 7.

But, obtaining the Access Address is complicated, and involves the intruder, waiting for empty data packets. Usually, the pattern is 1 followed by a sequence of zeroes. If we observe enough packets, we can obtain a good idea of the identity of the packets. At this point we consider many candidate packets with access address.

The next step is the find the CRCInit. The way CRC is calculated using a LFSR (Linear Feedback Shift Register) [https://en.wikipedia.org/wiki/Linear_feedback_shift_register]. This LFSR is filled with a CRCInit. The input is the data packet, which is XOR'd with the output of the LFSR and then fed back in. The value that is inside the LFSR at the end is the CRC, which we have obtained from the raw data, along with the bits of the data-packet. If we reverse the process, as outlined by Dominc Spill in his paper and add the data bits and CRC as inputs, we find the CRCInit. We get candidate values for multiple packets.

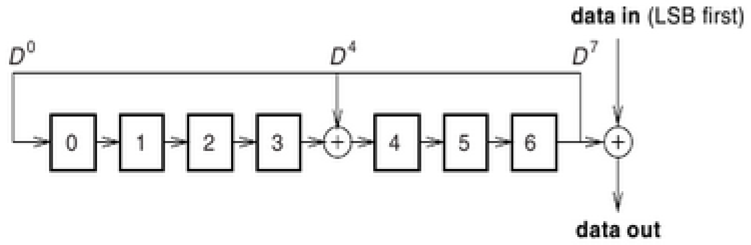


Figure 14 – CRC Calculation using LFSR.

The next step is to find the time slot length, which is the time between obtaining the data packet on the same channel . Channel hopping is performed from one channel to the other and keeps doing so for all 37 channels before it lands back on the same channel that it had started from. The trick is to keep sitting on one channel and then wait till we get back another data packet on the same channel.

If we find the time difference and divided it by 37, we get the time slot length.

$$\texttt{Time slot length} = \Delta t/37$$

The following step is to find the hop increment. We start on a data channel ‘n’ and wait till we receive a packet. Then we jump to data channel ‘n+1’ and we wait till we receive a packet. We then calculate the time difference between packet received on data channel ‘n’ and data channel ‘n+1’. We use hop-interval value and divide the wait time by this value to find the number of channels hopped between channel ‘n’ and channel ‘n+1’.

$$\text{‘n’} + \texttt{hopIncrement} \times \texttt{channelsHopped} = (n+1)\ (\bmod\ 37)$$

$$\texttt{hopIncrement} = \texttt{channelsHopped}^{-1}\ ((n+1)(\bmod\ 37))-n$$

$$\texttt{Using Fermats closed form : } \texttt{channelsHopped}^{-1} = \texttt{channelsHopped}^{37-2}\ (\bmod\ 37)$$

Using a Look Up Table, we find the inverse of channels hopped and obtain hopIncrement as an integer.

So we have: `Access Address , CRCInit , Time Slot Length and Hop Increment`.

With this , one can sniff new connections, sniff already-established connections, jam connections.

Current Bluetooth specification defines 3 security mechanisms- JustWorks which uses a temporary value of ‘0’ as a constant value. There is a PIN Exchange routing, that has a definite length limitation of only 6 digits which is not that hard to brute-force and OOB (Out of Band) which requires connection to the computer, display units and other IO peripherals.

The original PDU is encrypted using AES-CCM (WPA AES) algorithm. So there is essentially no way of getting the data values inside the PDU. The problem is with the Key Exchange Protocol.

In a usual eavesdropping scenario, User1 pairs with a brand new device. Intruder observes pairing/key exchange using Ubertooth. If JustWorks or 6 digit PIN, with good computational resources, Intruder obtains the Temporary Key (TK). With TK and pairing data, intruder recovers the Short Term Key(STK). With the STK and key exchange data that is being observed, intruder recovers the Long Term Key (LTK or session key). Each connection uses a different nonce, so the intruder actually has to witness the connection happen. This can be overcome by session jamming and forcing a reconnection for User1.

The way TK is calculated is, that p1 and p2 are exchanged over the air, with the random value.

```
AES(TK, AES(TK, rand XOR p1) XOR p2)
```

Which is confirmed on both the master and the slave. TK is a number from `0 to 999,999`, with the value being ‘0’ in JustWorks. The PCAP file is then decrypted using crackle tool [https://github.com/mikeryan/crackle] developed by Mike Ryan. The tool will decrypt a file with a pairing setup and an encrypted session using LTK.

From Vol. 3, Part H, Section 2.3.1 Bluetooth Core spec: "In LE legacy pairing, Authenticated man-in-the-middle (MITM) protection is obtained by using the passkey entry pairing method or may be obtained using the out of band pairing method."

From Vol. 3, Part H, Section 2.3.1 Bluetooth Core spec: "For LE Legacy Pairing, none of the pairing methods provide protection against a passive eavesdropper during the pairing process as predictable or easily established values for TK are used. If the pairing information is distributed without an eavesdropper being present then all the pairing methods provide confidentiality."

The passkey(s) are supposed to be random but it is not possible since the device does not have any IO capabilities. From Vol. 3, Part H, Section 2.3.5.3 Bluetooth Core spec: "The passkey Entry method provides protection against active “man-in-the middle” (MITM) attacks as an active man-in-the-middle will succeed with a probability of 0.000001 on each invocation of the method." This probability will increases on consecutive attacks, as passkey is static.

From Vol. 3, Part H, Section 2.3.5.3 Bluetooth Core spec: "The Passkey Entry STK generation method provides very limited protection against eavesdroppers during the pairing process because of the limited range of possible TK values which STK is dependent upon. If the attacker is not present during the pairing process then confidentiality and authentication can be established by using encryption on a future connection."

Spec version 4.0 Vol.3 p604 also reads , "Note: A future version of this specification will include elliptic curve cryptography and Diffie-Hellman public exchanges that will provide passive eavesdropper protection" , and as per Bluetooth 4.2, the pairing issues are being addressed by using ECDH. However, the current devices are on Bluetooth 4.1 and lower, including TI SensorTag.

The security is very important because devices such as security door lock, pace-maker , insulin pump among other medical control devices use this technology. . The injection mechanism is a bit more complicated but a reverse process data sniffing. Using link layer header and bad payload data, one can push that off to Ubertooth which calculates the CRC, whitens the data and pushes it out to CC24400. It is extremely complicated to do but with adequate resources, funding and motivation, it is not entirely impossible.

Chapter Four

Attribute Protocol and Attribute Profile

An attribute is 'data' entity that differs in context. It's could be data about the permissions, security requirements, addressable by a handle and has meaning elaborating its context. It's just like a remote-port, similar to a hardware register. It holds a 'value', addressed by a handle and UUID that describes what the attribute means. A device usually has more than one attribute so the address to the attribute is taken care of by the handle which is a 16bit value. It's not a memory address explicitly but can be considered one if you have a memory space, it could point to a memory address where the value is stored. It could be an abstract lookup table or a data register.

Data in the attribute can be hardware registers, device information, configuration information, command and control information etcetera. As it just exposes the data on a remote device, it exposes state. It is the most power efficient way to exchange small amounts of data.

There are two roles defined: Server role and Client.

Server role contains attributes, the information. It receives requests, executes on the request and responds. It can indicate values. The attribute server is allowed to also send responses without the requests. This is called notification.

Client role talks to server, sends out requests, waits or response, can confirm indications and carry out actions on the client system based on the received data values. The security algorithm is predominantly based on this exchange of data. The data itself is AES encrypted over the air interface.

Operations supported by the Attributes are:

- Push – Push some data value
- Pull – Pull the temperature of the environment
- Set – Set the name of the device.
- Broadcast - Send out a broadcast data, for example – if there is an earthquake, based on the sensor data of seismic activity, send out data on evacuation.
- Get – Get all the services rendered.

PUSH operation depends on the server side implementations of data transmission when it changes on the sensor. The client configures server to indicate the attribute, may configure triggers, it may also configure reliable confirmations, upon receiving the indication. This involves some aspects of enabling acknowledgement of indication and flow control using handle value indication.

PULL operation requests data from the server when it needs it. The polling is generally inefficient for data sets that are changing rapidly. This uses READ request or READ BLOB request.

SET operation can set attributes to configure a server, set the attribute to send the notifications. It may be used as actuators to control devices. It uses WRITE request or WRITE BLOB request. It is a complete client side implementation. There is a Prepare Write Request / Execute Write Request where we prepare a Write request and push it to server that sends back a query to the client to confirm the requested WRITE operation. On reception this information, the execute WRITE Request is executed.

BROADCAST operation is a standard format with a length:tag:data. It is configurable on the server and can be autonomous.

GET operation enables the client to get the attributes of the device with their UUIDs and handles. This proves that the configured UUID service exists on the device and also, find specific types of a device. It uses READ information request or READ by UUID request.

A complete list of the client / server protocol list is given below.

Client	**Server**	**Notes**
	Error	The request sent failed
Read	Read	Used to get attribute handles
Read By	Read	Used to get an attribute type and pull value
Read	Read	Used to pull the value of an attribute
Write	Write	Used to set the value of an attribute
Read Blob	Read Blob	Used to pull a long value of an attribute
Write Blob	Write Blob	Used to set a long value of an attribute
Read	Read	Used to do an atomic read/write

Prepare	Prepare	Used to prepare a set of writes
Execute	Execute	Used to execute the prepared writes
Handle Value Confirmation	Handle Value Indication	Used to push values and receive confirmations.

An Attribute Profile is the flat structure that is a collection of attribute. These attributes may have different data values and contexts. These attributes need to be grouped into a collective that have a definitive context. These collectives can be described as a service.

The following example will bring more clarity to the case.

Attribute	**Value**
Service	GAP
Device Name	“Gune’s Device”
Service	Attribute Profile
Attributes Changed	0x0000
Service	ID Profile
Sub-Service	Time
Current Time	22:17
Client	0x000000000000
Update interval	10 mins
Service	Proximity Profile
In-alert	0
Out-alert	0
Tx power	4

This currently looks flat and does not depict the hierarchy that makes any sense out of this structure of data.

When we add a layer of demarcation according different services

Attribute	Value
Service 1	GAP
Device Name	“Gune’s Device”
Service 2	Attribute Profile
Attributes Changed	0x0000
Service 3	ID Profile
Sub-Service	Time
Current Time	22:17
Client	0x000000000000
Update interval	10 mins
Service 4	Proximity Profile
In-alert	0
Out-alert	0
Tx power	4

With sub-service level demarcation and characteristic specific demarcation, we have

Attribute	Value
Service 1	GAP
Device Name	“Gune’s Device”
Service 2	Attribute Profile
Attributes Changed	0x0000
Service 3	ID Profile
Sub-Service	Time
Current Time	22:17
Client	0x000000000000
Update interval	10 mins
Service 4	Proximity Profile
In-alert	0
Out-alert	0
Tx power	4

Attributes on a server have permissions that can only be discovered and access if the API level permissions on the client device are validated. An attribute protocol request may very well fail if there is a failure with the authentication. Once the authentication is successful and the device is connected, the attributes within the group “service” will be authenticated and all sub groups will have visibility.

A meta-level implementation of attribute is called a meta-attribute that describes what the parent attribute can do: READ, WRITE, NOTIFY. The context metadata is described in the meta-attribute. It provides a grouping scheme in the represented data format:

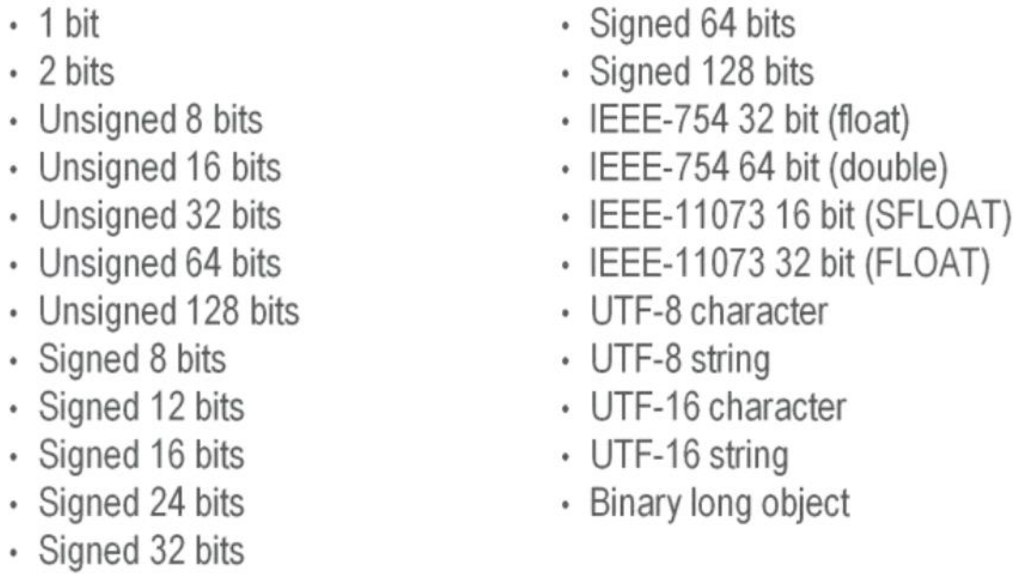
Meta Attribute – “Data Dictionary, Readable, 10^{-2}”

Auxiliary Data – “°C”, “Outside”

Configuration – Enable Indications

Triggers – Indicate x>25 °C OR x<16 °C

This is implemented in order to have a universal profile set up, that does not require client side UI specific implementation to read and make sense of the data. The meta-data defined in the meta attribute will sufficiently serve the context of the sensor data that is either explicitly pulled or notified. The data dictionary is type of data that usually describe the type of the attribute in units or physically defined standards followed by all and are re-used. All of these are represented by a 16-bit UUIDs. Data dictionary describe the type of value but does not define the format of the data. The meta-attribute also has a data-types and exponentials for formatting.



- 1 bit
- 2 bits
- Unsigned 8 bits
- Unsigned 16 bits
- Unsigned 32 bits
- Unsigned 64 bits
- Unsigned 128 bits
- Signed 8 bits
- Signed 12 bits
- Signed 16 bits
- Signed 24 bits
- Signed 32 bits
- Signed 64 bits
- Signed 128 bits
- IEEE-754 32 bit (float)
- IEEE-754 64 bit (double)
- IEEE-11073 16 bit (SFLOAT)
- IEEE-11073 32 bit (FLOAT)
- UTF-8 character
- UTF-8 string
- UTF-16 character
- UTF-16 string
- Binary long object

Figure 15 – Data Types Supported

Obscure data types such as IEEE-11073 16 bit (SFLOAT) and IEEE-11073 32 bit (FLOAT) are predominantly used in medical profiles, which makes it possible for application developers to port these attributes into their custom application.

Example :

```
MetalAtrribute(Temperature_DD, Signed Integer 16 bit, -1)

{

       ATTR_VAL = 225;
       DESCRPTION = “Temperature”;
       POSTFIX = ‘°C’;
       User_Info = “Outside”;
       //-1 is assumed to be the exponent and renders ATTR_VAL as 225 x 10$^{-1}$

}

The client UI reads: Outside Temperature 22.5°C
```

Triggers are executed on operations like:

< , > , <= , >= , = = , ! =, changed, not-changed,! (indication) in given time.

If the data change observed is too often, the server stores values and sends it out slowly in batches. This is done with a first-in-first out buffer (FIFO). The order is maintained, and thus allows storage of values between connection events.

The attribute profile also supports meta-meta-attributes, which is called aggregation. This allows us to associate two pieces of information in one value. For example,

```
Meta-Meta-Attribute(TimeStamp, Temperature){

       ATTR_VAL = 20090228:24;
       TimeStamp: Meta-Attribute(Clock,blob,0);
       Temperature: Meta-Attribute(Temp,sint16,-1);

}

Which on the UI translates into:
On 28th February 2009, the temperature was 2.4°C
```

Chapter Five

Texas Instruments : SensorTag Overview

Overview on TI SensorTag

Texas instruments CC2640 is a low power consuming, coin cell battery operated sensor-development platform for developing single-mode BLE applications. It is based on SimpleLink CC2640 System-on-Chip (SoC) Bluetooth Smart solution. It combines 2.4GHz RF transceiver, 128kB of in-system programmable memory, 20kB of SRAM and some peripherals. It has an ARM Cortex-M3 series processor that handled all the application layer and BLE protocol stack, autonomous radio core based on ARM Cortex-M0 which deals with low-level radio control and processing associated with the physical layer and parts of the link layer.

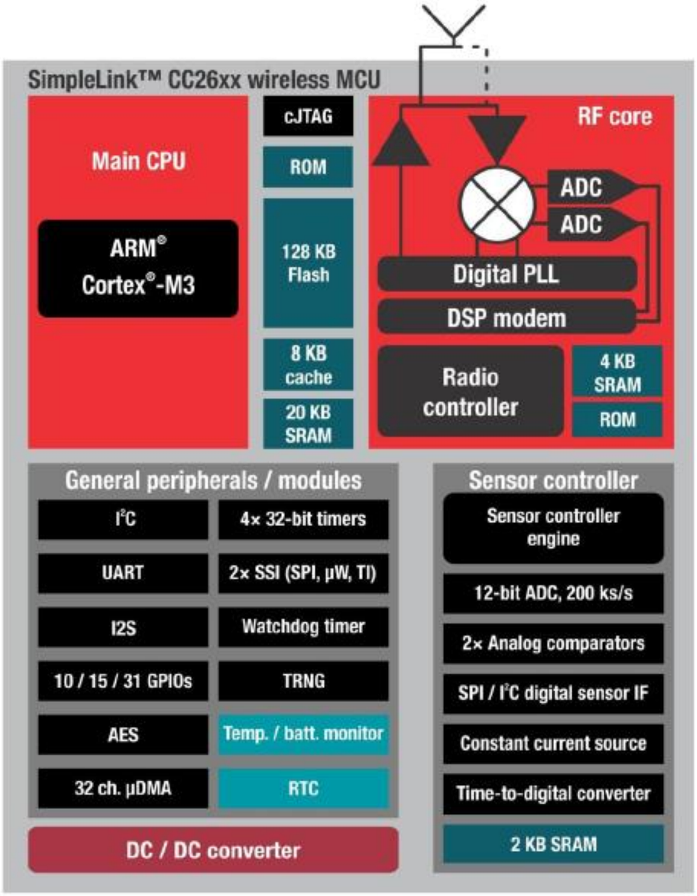


Figure 16 Block Diagram of TI CC2640

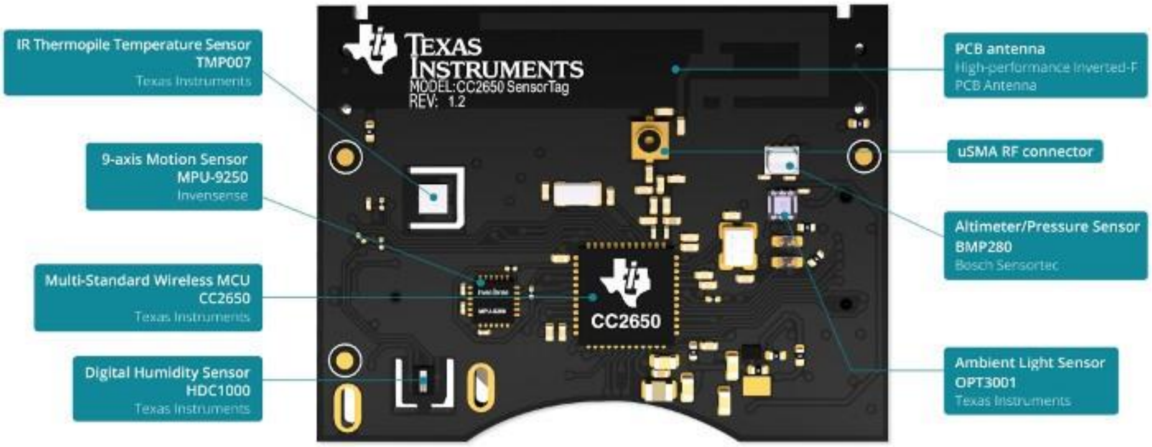


Figure 17 – Top view of TI SensorTag

The most cheapest and low-power using configuration of usage for this SensorTag is the single-device solution that has all the controller, host, profile and application implementation.

It is composed of Real Time Operating System (TI-RTOS with SYS/BIOS kernel), CC26xx driver libraries, BLE protocol stack and some sample applications and profiles that are fully qualified by Bluetooth SIG.

It contains a SensorTag project in the BLE stack SDK which is configured to run on CC2650 SensorTag hardware platform and is only able to communicate with SensorTag's on-board peripherals.

The software architecture is composed of the TI-RTOS which is a real-time preemptive, multi-threaded operating system that runs the software solution and synchronizes tasks. The BLE stack and Application exist as separate entities on the RTOS. The stack image includes lower layers of BLE, from Link Layer up to GATT layers, including the GAP layer as well. The Application layer includes profiles, application code and ICALL module that is used to communicate between the application RTOS thread and BLE stack RTOS thread.

As the application image is different from the stack image, the upgrade of the application is much easier from a developer's point of view. This is the result of the Indirect Call Framework (ICall). It is a mechanism for the application to interface with the BLE protocol stack services (BLE Stack APIs) as well as thread synchronization provided by RTOS.

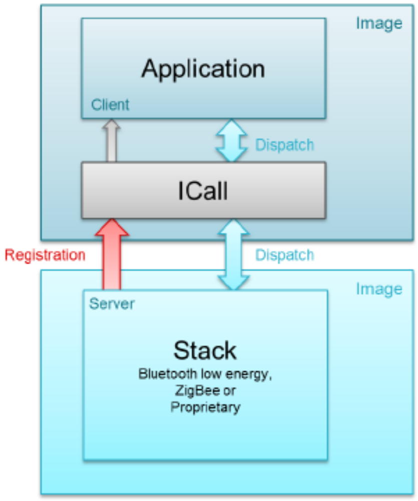


Figure 18 ICall Application to Protocol Stack abstraction

Two tasks can send each other a block of message from one thread to the other via a message queue. The sender allocates memory, which is due to the global heap APIs for dynamic memory (HEAPMGR_SIZE preprocessor define in the application project) and writes the message into the memory block enqueues it in the message queue using a signaling semaphore. The receiver wakes up on the semaphore, copies the message block, processes the message and frees the memory block to the heap.

Over the Air Download

TI's SensorTag supports Over the Air download option for firmware on the SensorTag system. The target device does not need to be physically present or accessed to provide an application software upgrade. It involves providing some instructions that enable OAD in an application project, creating an image, verifying it's correctness, sending out the application Over the Air using either an Android or an iOS implementation of the OAD feature and having it re-verified on the target device before running it from the bootloader.

It is necessary to mention that enabling OAD is currently designed to work with IAR software and CC2640 (and up) series devices. One cannot create OAD images using TI's open source integrated development environment called Code Composer Studio which uses TI's Compile.

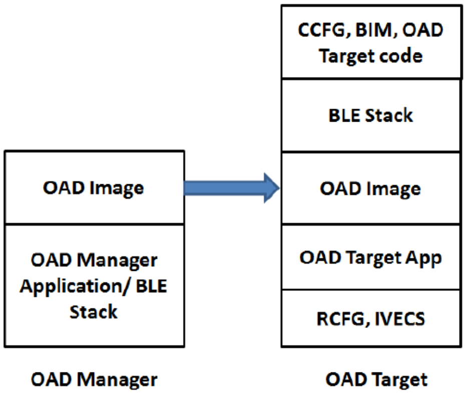


Figure 19 OAD Manager and OAD target block diagram

The OAD manager contains the image that will be sent over the air and an application that performs the server role of the OAD process. There are two OAD images that exist simultaneously on the system on chip. Only OAD Image B can be downloaded, onto a specific area. The Boot Image Manager, holds the permanent boot code that provides a fail-safe mechanism for determining which image is ready to run. It searches for a valid image and jumps to that image for execution. Either Image A or B must implement the proprietary TI OAD profile. Image A reflects this role by default. The BLE Stack image is shared by the OAD Target App and OAD image. It can never be upgraded or modified via OAD. It is a hex file that is merged with the OAD Target App and BIM.

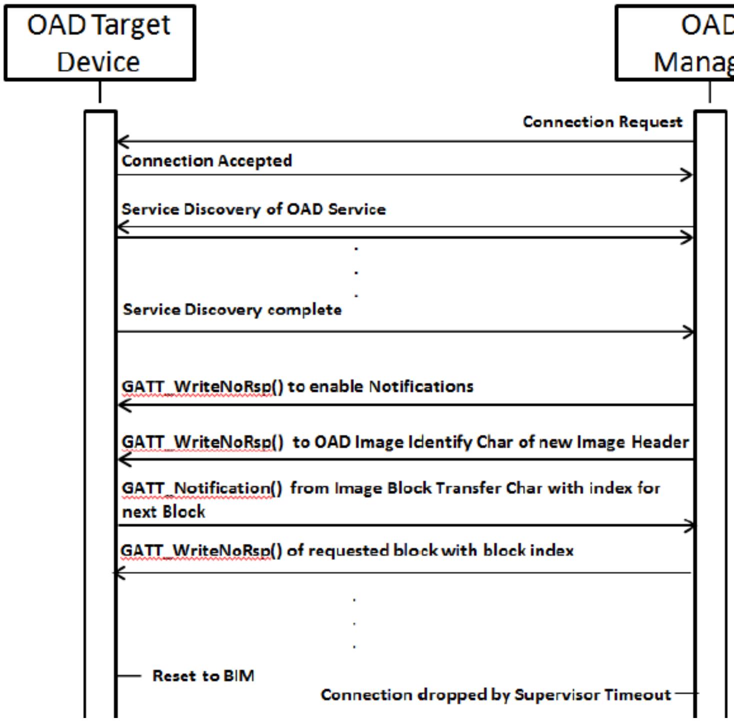
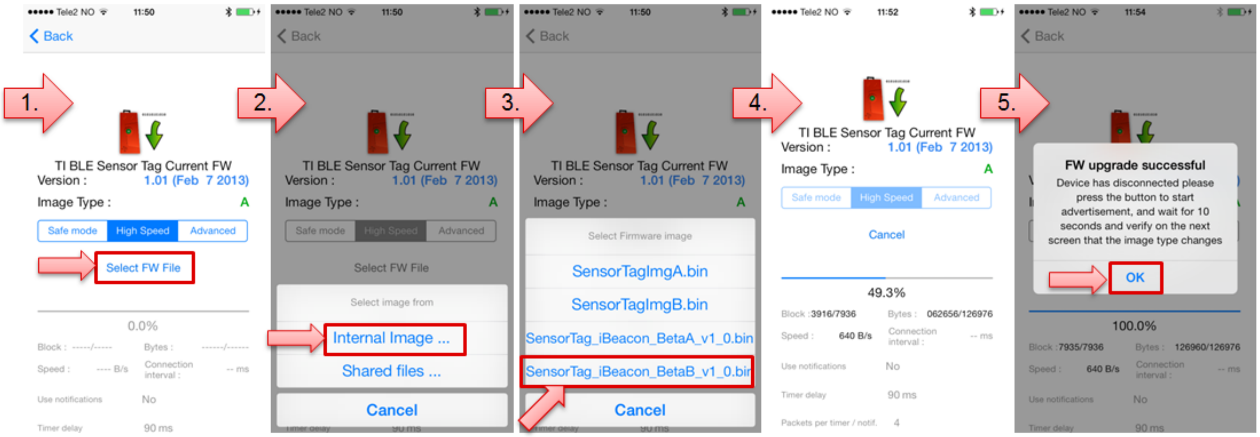

OAD Target Device
OAD Manager
Connection Request
Connection Accepted
Service Discovery of OAD Service
Service Discovery complete
GATT_WriteNoRsp() to enable Notifications
GATT_WriteNoRsp() to OAD Image Identify Char of new Image Header
GATT_Notification() from Image Block Transfer Char with index for next Block
GATT_WriteNoRsp() of requested block with block index
Reset to BIM
Connection dropped by Supervisor Timeout

Figure 19 OAD procedure

OAD process in iOS App



1. TI BLE Sensor Tag Current FW
Version : 1.01 (Feb 7 2013)
Image Type : A
Safe mode | High Speed | Advanced
Select FW File
0.0%
Block : -----/----- Bytes : ------/------
Speed : ---- B/s Connection interval : -- ms
Use notifications No
Timer delay 90 ms

2. Select FW File
Select image from
Internal Image ...
Shared files ...
Cancel

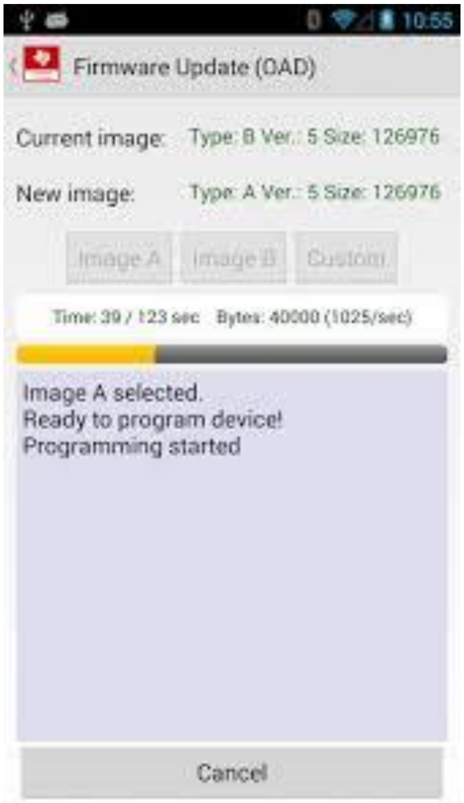
3. Select Firmware image
SensorTagImgA.bin
SensorTagImgB.bin
SensorTag_iBeacon_BetaA_v1_0.bin
SensorTag_iBeacon_BetaB_v1_0.bir
Cancel

4. TI BLE Sensor Tag Current FW
Version : 1.01 (Feb 7 2013)
Image Type : A
Safe mode | High Speed | Advanced
Cancel
49.3%
Block :3916/7936 Bytes : 062656/126976
Speed : 640 B/s Connection interval : -- ms
Use notifications No
Timer delay 90 ms
Packets per timer / notif. 4

5. FW upgrade successful
Device has disconnected please press the button to start advertisement, and wait for 10 seconds and verify on the next screen that the image type changes
OK
100.0%
Block :7935/7936 Bytes : 126960/126976
Speed : 640 B/s Connection interval : -- ms
Use notifications No
Timer delay 90 ms

Figure 20 – iOS implementation of Firmware update Over the Air

OAD process in Android App



Current image: Type: B Ver.: 5 Size: 126976

New image: Type: A Ver.: 5 Size: 126976

Time: 39 / 123 sec Bytes: 40000 (1025/sec)

Image A selected.
Ready to program device!
Programming started

Figure 20 Android OS implementation of Firmware Upgrade

This presents a new challenge to the developers and users of TI's SensorTag as anyone with the SensorTag application can connect to the device, and update the firmware and render the target device un-usable, corrupt and can even be up to malicious intent. In fact, being able to change the firmware presents evildoers with many different opportunities to cause great loss to people, businesses and processes that could end up catastrophic. My proposed solution attempts to create an authentication procedure that requires credentials to be able to have access to upgrade the firmware, and since the resources on the device itself are scarce, most of the computational burden shall be taken up by the software application custom created on the Android operating system.

Chapter Six

Proposed algorithm for Authentication and Authorization for OAD FW Updates

As discussed in the closing remarks of chapter five, any person with a smart-phone and a successfully built 'FWImage.bin' image file can potentially update the firmware of the device remotely using a cloned version of Texas Instruments app on the Android and iOS platforms. This causes huge concerns over security and feasibility of the device in use and could end up being very catastrophic to the users and the industry as a whole.

My solution takes leverage of the rich operational and computational resources on the client app implementation on Android OS and reduces computational burden on the device, so as to obtain a secure and verified way to update firmware, only intended by the authorized app, so as to mitigate the problems that are caused by the absence of security measures that are observed in current implementations of Over the Air firmware update feature offered by Simplelink's TI SensorTag app on Android OS. These concerns may also be mimicked by the iOS implementation as well, but I have not studied the iOS source code for the application to be certain of the same. The algorithm however, is universal and is simple yet robust. It's application can be ported and implemented to iOS as well as WinRT platforms with relative ease, and depends on the richness of the Bluetooth LE API set provided by SDK's associated with iOS and Windows Phone/ Tablet platforms respectively.

The proposed algorithm involves firmware modification and hence creation of a custom service with customized data attributes defined by their characteristics that hold data values. This solution behaves universally to all applications that have generic GATT API implementations but will only work with applications that have custom code on the client side that defines and responds to behavior pertaining to the implementation of the algorithm.

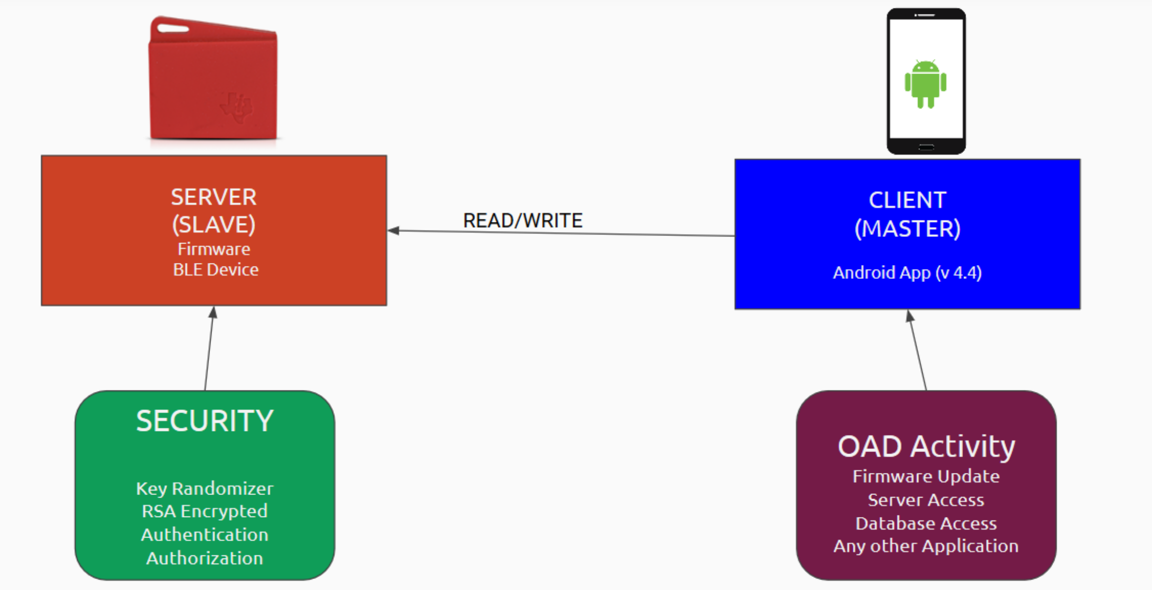


Figure 21 : The General Idea

The algorithm is a simple implementation of READ and WRITE functionality offered by the BLE API to combine current encryption and cryptography standards and create a new custom authentication and authorization service that is robust and effective.

I have developed a new custom service called Security Service, that contains three characteristics, which have a READ , READ/WRITE and READ operation permissions.

These hold data values that can only be verified and written by the correct context that is defined in the client side application. The computation involved on the device is very minimal. Only the RSA encryption and Phase 2 decryption mechanism is applicable on the device. This is designed to reduce the load and power-consumption on the server side.

The basis of encryption takes advantage of the fact that generating/choosing prime numbers is easy, multiplying two numbers is easy but finding the prime-factors of a given number is hard. For a given number 'n', finding the prime-factors p and q is quite difficult. The fastest methods still take an order of (n raised to 0.5) steps.

If we consider n to be the product of two prime numbers; m and c to be integers between 0 and n-1 and e be an odd integer between 3 and n-1 that is relatively prime to (p-1) and (q-1), given n,m and e , it becomes easy to compute m^e(mod n).

m^e (mod n) is the result of multiplying e copies of m, dividing by n and keeping the remainder. The reverse of this is also easy, given the prime factors. If we have n, e, c and prime factors p and q, it is easy to recover the value of m, where m = $(m^e)^d$ mod n. The integer d is easy to compute if we have e, p and q. From a security perspective, if we only have n, e and c but not p or q, it becomes very hard to extract the value of m. These observations formed the basis of the RSA cryptosystem, developed by Ronald Rivest, Adi Shamir and Leonard Adleman in 1977.

The public key in this system is value of n (also called the modulus), while the value of e is called the public exponent. The private key pair consist of n and value of d, also known as the private exponent.

The steps to generate the public-private key pair are :

1. A pair of random prime numbers.
2. Computation of modulus : n = pq
3. Selection of an odd public exponent e , such that 3<e<(n-1) and is relatively prime to (p-1) and (q-1).
4. Computation of private exponent d, from e, p and q.
5. Private Key is generated using (n,d) while public key is generated using (n,e).

Which boils down to :

$$\text{Message} = m$$

$$c = \text{ENCRYPT}(m) = m^{e} \pmod n$$

$$m = \text{DECRYPT}(\ c\ ) = c^{d} \pmod n$$

Implementing a Diffie-Hellman model for message verification, a message can be digitally signed by applying the decryption operation to it and verified using the encryption operation, as described below.

$$s = \text{SIGN}(m) = m^{d} \pmod n$$

$$m = \text{VERFY}(s) = s^{e} \pmod n$$

The secrecy of the $\phi(n)$ is ensured by prime factors p and q.

$$\text{Totient function offers : } \phi(n)=(p-1)(q-1)=pq-p-q+1=(n+1)-(p+q)$$

$$(n+1)-\phi(n)=p+q$$
$$(n+1)-\phi(n)-p=q$$

$$\text{and } n = pq$$

$$\text{hence, } n = p(n+1-\phi(n)-p) = -p2+(n+1-\phi(n))p$$

$$\text{which gives us, } p2-(n+1-\phi(n))p+n = 0$$

$$\text{which is a quadratic equation in p with } a = 1 \text{ , } b = -(n+1 - \phi(n)) \text{ and } c = n$$
$$\text{solving for p gives us}$$
$$p = (\ -b \pm (|b|^{2}-4ac)^{0.5}\ ) / 2a$$

substituting,

$$((n+1 - \phi(n)) \pm((|(n+1 - \phi(n))|^{2}-4n)^{0.5}))/2$$

Considering an example : let $n = pq = 377$ such that $p = 13$ and $q =29$
$\phi(n) = (p-1)*(q-1) = 336$
$a =1$, $b = -(377+1-336) = -42$, $c=377$
$p^{2}-42p+377 = 0$, hence $p = (26/2)$ and $q = (58/2)$ which makes $p=13$ and $q=29$.

Generating the private exponent involves using the extended Euclidean algorithm, which is the basic Euclidean algorithm that we use for GCD, but ran backwards.

We intend to find the value of d, such that $e * d = 1 \pmod{\phi(n)}$

The EED calculates x and y such that : $ax + by = gcd(a,b)$; $gcd(e, \phi(n)) = 1$ by definition.

$$ex+ \phi(n)y = 1$$

$$\text{therefore , } ex = 1 \pmod{\phi(n)}$$

$$x = d$$

The Algorithm : GuneSec

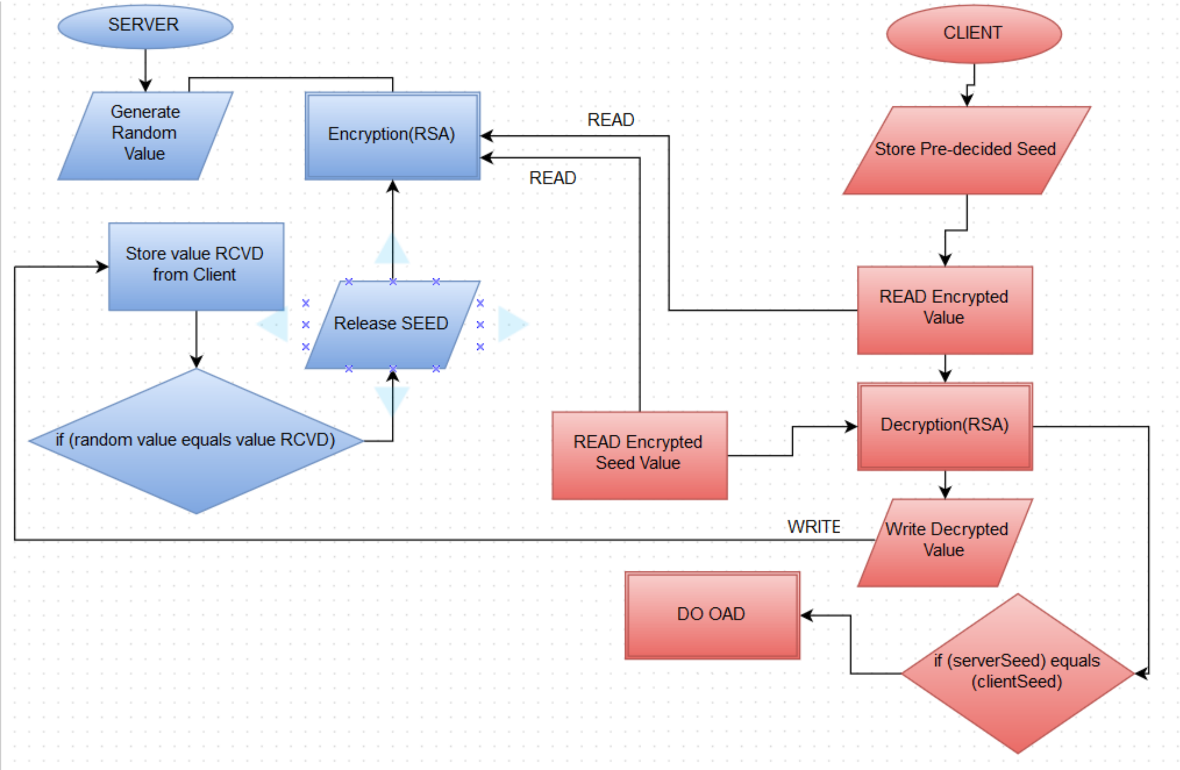


Figure 22 – The Algorithm

This algorithm implements the authentication and authorization measures needed to have the ability to perform OAD Firmware Update.

A Seed value is predefined by the Server and the Client during the manufacturing stages so that it forms as a credential for authentication.

As soon as the client is connected to the BLE device, it polls for random number generation. The server(BLE Device) generates random number. One of these numbers is used as a key and is encrypted using the RSA module. This value is then read by the client. As the READ function is carried out over the air interface, the key is secured due to the native AES CCM encryption found in Bluetooth Low Energy Spec. After receiving this value, the client uses its decryption module to obtain the decrypted value. Upon successful decryption, it writes back the decrypted value to a data-holder on the Server. Then, the client READS the flag variable. This READ request causes the BLE device to verify the previous response sent by the client. This verification, if successful, releases an encrypted version of the seed. This is then READ by the client, and acts as Phase 2 of the authentication mechanism. The client then manually decrypts the received value and enters the seed value. If the entered value is the same as the one that existed on the client, the OAD FW update module is offered for use.

FIRMWARE IMPLEMENTATION

The firmware exists on the SensorTag and holds all the necessary variables as well as verification, encryption and seed disclosure modules. It has security service configured which is invoked by the client . Custom UUID's help define and recognize service elements. The Services help defines Characteristics which offer data values to store and retrieve along with READ/WRITE/NOTIFY properties. It also has a parent C file called SensortTag.c that implements the services, as well as carries device information and enables the Over the Air Download Module.

The custom service I have defined is..

- Modeled in the GATT (Generic Attribute Profile) paradigm
- Sits on top of the ATT (Attribute Protocol)
- Functions in the SERVER role as a SLAVE.

Some variables and constants are declared to keep up with the design requirements outlined in the Bluetooth LE Specification.

UUID Definition :

```
MW Base 128-bit UUID: 3E09-[HI][LO]-293F-11E4-93BD-AFD0FE6D1DFD

MW_BASE_UUID_128( uuid )
  0xFD, 0x1D, 0x6D, 0xFE, 0xD0, 0xAF, 0xBD, 0x93, 0xE4, 0x11, 0x3F, 0x29,
LO_UINT16( uuid ), HI_UINT16( uuid ), 0x09, 0x3E.

SECURITY_AddService- Initializes the SECURITY service by registering GATT
attributes with the GATT server.
```

bStatus_t `security_AddService`(uint32 services);
This is called in by the SensorTag.c file to register the said service.

UUID Declarations :

```
SECURITY_SERV_UUID  0x9923 // Service UUID

PUBLIC_DATA_UUID    0x9915 // holds the public key

RCVD_DATA_UUID      0x9916 // holds the rcvd_secret

FLAG_DATA_UUID      0x9917 // holds the flag value; if (TRUE){ value = seed }

SECURITY_DATA_UUID  0x9919 // Random Value Storage
```

Services in SensorTag (BTLE) :

- C array containing pre-defined constants
- Contain Characteristics linked to UUIDs , Size and Type and Permissions.
- Function calls to register them.
- This is called creating an Attribute Table.
- These services are registered in master <SensorTag.c>

```
static gattAttribute_t securityAttrTable[] =
{
 { // Service declaration
       {

     ATT_BT_UUID_SIZE,

     SecurityServiceUUID }, /* type */

     GATT_PERMIT_READ, /* permissions */

     0, /* handle */

     (uint8 *)&ServiceValue /* pValue */


   },
```

The service is declared in the C-Array. gattAttribute_t is the callback function that offers the structure to declare the service desired. ATT_BT_UUID_SIZE is the constant 128-bit UUID size used pan-Bluetooth LE. The permissions declare what kind of activity is offered on the given service. The handle provides various values that offer certain kind of READ/WRITE modes defined by Bluetooth Spec. I've used 0, which is default mode.

```
// Characteristic Declaration
    {
      {
       ATT_BT_UUID_SIZE,
        characterUUID
      },
        GATT_PERMIT_READ,
        0,
        &SecurityDataProps
    },
```

Within the Attribute table definition, the characteristic is declared. The characterUUID is the variable that holds the UUID that is defined in the <service.h> file. The reference to SecurityDataProps is just a placeholder variable that details the kind of properties offered to this characteristic. I've not explicitly defined any property to this characteristic.

```
// Characteristic Value "Data"
   {
     {
      MW_UUID_SIZE,  //UUID
       SecurityDataUUID

      },

      GATT_PERMIT_READ |
      GATT_PERMIT_WRITE,
       0,
       SecurityData
   }
```

The third element defined in the array, is the actual data-value holder for the said characteristic. SecurityDataUUID points to the UUID defined in the <service.h> file. The READ/WRITE properties defined above grant the ability to READ the value existent or WRITE to the characteristic from the client.

API calls :

- 3 Permitted Callbacks : READ , WRITE and NOTIFY.
- I have implemented them for R/W modes.
- The READ/WRITE callback executes the UUID switch case
- Which parses the data object into memory (memcpy).

```
// Register GATT attribute list and CBs with GATT Server App


status = GATTServApp_RegisterService

(

      SecurityAttrTable,

     GATT_NUM_ATTRS( SecurityAttrTable),

     &SecurityCBs );

}
```

The GATTServApp registration service is overloaded with SecurityAttrTable that has the attribute declarations for characteristics. The GATT_NUM_ATTRS macro counts the number of attributes present in the service definition array. SecurityCBs are callback functions that declare and register the Read and Write Callback functions.

```
CONST gattServiceCBs_t SeurityCBs =
{

  Security_ReadAttrCB,  // Read callback function pointer

  Security_WriteAttrCB,  // Write callback function pointer

  NULL                   // Authorization callback function pointer

};
```

The gattServiceCBs_t offers structures to define custom callback functions that aid in the READ/WRITE functionality. The above code snippet declares two callback functions, Security_ReadAttrCB and Security_WriteAttrCB.

```
static uint8 SecurityReadAttrCB (args)

{

    uint8 uuid;
    bStatus_t status = Success;
 switch(uuid)
{

    case UUID:
    memcpy(pValue,data[0],LEN);
    { do what you want
      with data[0]

}
    break;
    return(status);
}
```

The callback functions offer cases based on the UUID. Each case deals with the operation that is offered in the given callback function. pValue is the data buffer that holds the value that is calculated by the switch case. It is stored in the memory using the memcpy() routine. The status variable manages the state machine logic and helps the handover of the threads from the GATT_Service to GAP_Role.

READING

- The Bluetooth LE Stack receives the READ request from the client, over the air.
- The stack treats this request as a GATT_MSG_EVENT.
- The GATT_MSG_EVENT is then sent to GATTServApp routine.
- This GATTServApp routine triggers the execution of the profile callback and the associated READ functionality according to the UUID and the permission.
- Memcpy executes as the value is copied to the memory. This value is returned to GATTServApp.
- GATTServApp returns this value to the Stack that returns the response to the GATT Client, over the air.

WRITING

- The Bluetooth LE Stack receives the WRITE request with a value to be written, over the air.
- This is treated as a GATT_MSG_EVENT.
- The GATT_MSG_EVENT is then sent to GATTServApp routine.
- This GATTServApp routine triggers the execution of the profile callback and the associated WRITE functionality according to the UUID and the permission.
- Profile stores the value, then writes it to the respective characteristic data-holder.
- Then the Profile notifies the app by calling message enqueuer callback and posts its semaphore using the ICall Framework.

ANDROID (CLIENT) IMPLEMENTATION

The Google API provides a rich set of API's to request data from the Bluetooth Device and Write data to the Bluetooth Device. It also offers a host of neat modifiers and a library of other API's that help communicate with the web and bring Internet of Things alive.

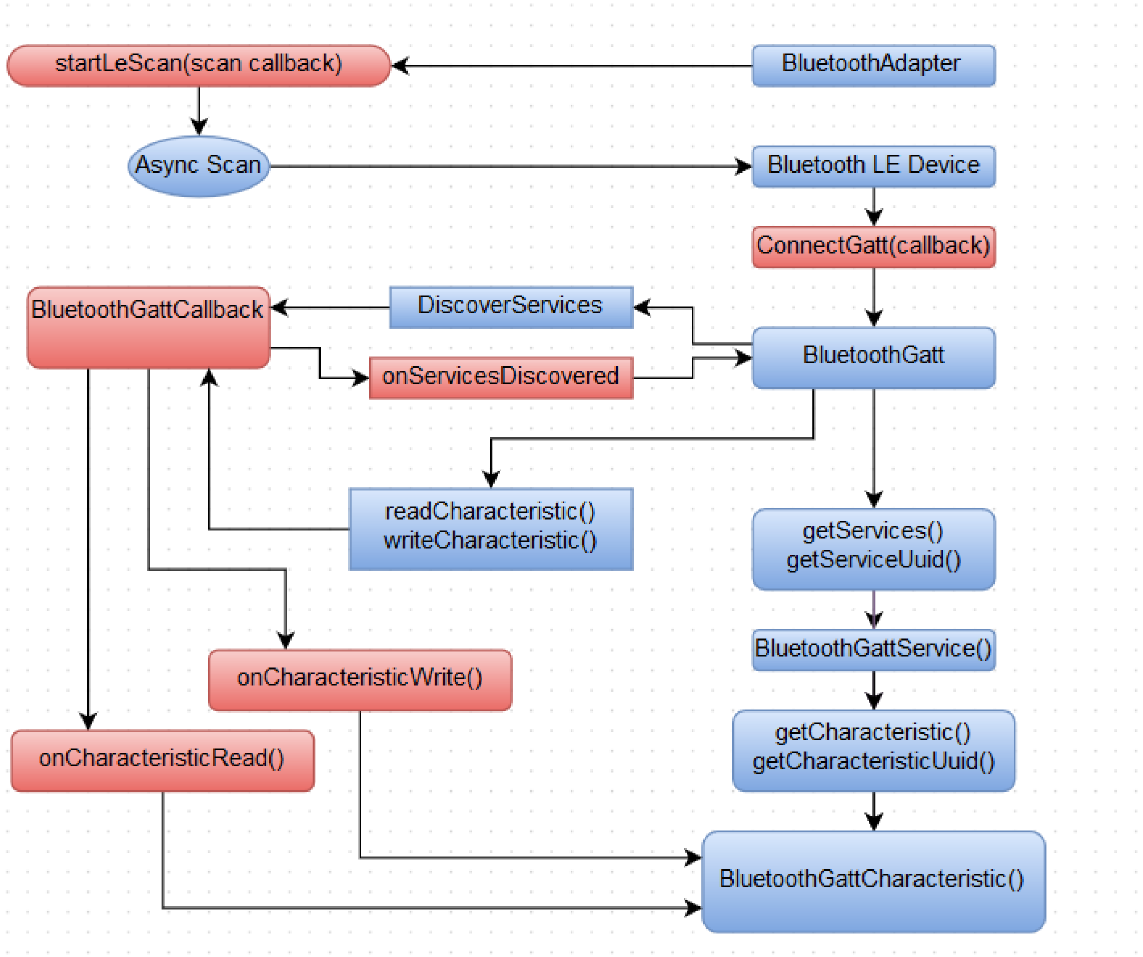


Figure 23 Android Flow

The client has a native Bluetooth Adapter that helps initialize Bluetooth on the client, and performs scanning for nearby Bluetooth LE devices. This scan is asynchronous. Once a device is found, a connect request is sent to the LE Device.

Upon successful connection, the BluetoothGatt API is invoked to perform discovery of Services on the LE Device. This invokes a BluetoothGattCallback API that has many Callback functions that execute the desired functionality.

After the services are discovered, the BluetoothGatt object requests reading of a characteristic defined on the said service or writing of a value to the said characteristic. This is set up by finding out all the services present on the device and then selecting the desired one by identifying it's UUID.

The BluetoothGattService API offers other functions that help get Characteristic specific functions. Once the intended characteristic is found using it’s UUID, a readCharacteristic() method is invoked.

This offers the ability to read the value using getValue() method within the Characteristic that is just read. On the flipside, if a Characteristic needs to be written a value, the value must be first converted into byte format and needs to be passed as an argument to setValue() method. This is then passed into writeCharacterstic() method.

Both – readCharactersitic() and writeCharacteristic() fire callback functions onCharacteristicRead() and onCharacteristicWrite(). On successful execution, the value is either written or read. If the function is not successful, the program notifies the user of the failure. If there is a null value passed in any of the variables or pointed to by any object, the program crashes.

Chapter Seven

Future Tests and Extensions

POWER CONSIDERATIONS

As the Internet of Things is purely based on the ability of the deployed devices to work for a long time, the need for low power consumption is quite important. This required an analysis of power consumption for SensorTag in different modes with different sensors in on and off state.

- When the device is not connected to client and is in its Stock firmware (LED Blinking) : 0.35 ~ 1.24 → 2mA(max)

- When the device is connected to client app on Android still using Stock Firmware : Connecting to SensorTag App : 4.7mA(max)

- When all of the sensors on the Device are shut OFF: 0.50 mA

- When Only Motion Sensor (Accelo,Gyro,Magneto) is turned ON : 4.20mA

- When only Humidity+Temp Sensor is kept ON : 0.76mA

- When the data rate is increased to MAX - Fast Data Rate : 1mA

- All sensors ON, max refresh rate : 5.28mA

After these preliminary tests, I decided to take 3 different batteries at different voltage levels.

Tests done with 3 batteries(2.6V, 3.0V, 3.06V)

- When the Firmware is only configured to shoot out a single beacon at about 5 second time interval ONLY BEACON FW : Does not work for below 3.0V

- Stock SensorTag FW : Does not work for below 3.0V

- Depreciation for Beacon FW : 3.06V → 2.6V in 2 weeks.

In Modest conditions @ 2.0mA , it will still only last : 120 hrs of runtime.

This is a matter of concern. As per the Bluetooth Specification, the energy transaction is well designed to work with coin-cell batteries.

As the maximum transaction time is 3ms, the transmit power being 15mW if we assume the usage of 40nm chips ,

For a 1.5 V battery offering a 10 mA current , the energy per transaction =

```
                  0.015 W x 0.003 Sec = 45 mJ (micro Joules)
```

Considering a 1.55 Volts battery at 180mAh @ 10mA per transaction, sold for about $2 , the number of transactions that could be theoretically done are :

```
180 mAh / 10 mA = 18 hours = 64800 seconds = 21.6 Million transactions.
```

The Bluetooth Specification has designed the Low Energy version to really consume very little energy and sleep for durations when not in use.

This means that more research needs to be done at the power consumption for TI's SensorTag.

EXTENSIONS

The algorithm is designed for all devices supporting Bluetooth Low Energy. The implementation at this time is only restricted to Android taking on the Client Role and TI's SensorTag taking the role of the Server (Peripheral).

The next step is to implement the algorithm in iOS and WindowsPhone/WindowsRT devices, as they offer their own set of API's.

Future tests also include other BLE capable devices that are out on the market or are currently in beta testing phase, awaiting deployment.

The Diffie-Hellman implementation was also alluded to in one of the sections on the firmware but was never implement as it broke the functionality and hence was left out. It needs to be tested thoroughly. This forms as a motivation amongst other encryption standards.

CHALLENGES

- Testing was hard : Unstable behavior of stack. This was due to the fact that the connection only lasts for 3ms (transaction time) and hence, if the data is not constantly polled, the connection to the device is lost.

- Using higher key-values for RSA breaks thread behavior : FW and Android are not able to computationally calculate exponentials in greater than 20.

- Setting constants for characteristic data causes compile errors. This was due to the fact that they would occasionally conflict with the callback functions.

- Invoking API calls that trigger callbacks was tricky/complicated.

- Android OS behavior needs context and view declarations that is a bit hard to implement for those who are new to

- Threading issues for different views as the BluetoothLE thread is different from the User Interface thread and this causes clashes and null-pointers causing the program to crash.

- Calling data-variables within scope of FW, causes undesired callbacks as each variable is basically a characteristic and hence executes callbacks associated with the said characteristic.

- Loops do not work for READ/WRITE Callbacks as the calls are static and don't execute nested threads.

Resources

Firmware Code Repo :
https://github.com/shreyasgune/SensorTagFirmware

Android Client App Repo :
https://github.com/shreyasgune/Android-SensorTag-Security

Ubertooth :
https://github.com/greatscottgadgets/ubertooth

TI Resources :
Softwares and Stacks : http://www.ti.com/tool/sensortag-sw

Google :
http://developer.android.com/guide/topics/connectivity/bluetooth-le.html
https://github.com/googlesamples

Bluetooth Low Energy: The Developer's Handbook 1st Edition by Robin Heydon

Getting Started with Bluetooth Low Energy -Tools and Techniques for Low-Power Networking by Kevin Townsend, Carles Cufí, Akiba, Robert Davidson

http://www.bluetooth.com/what-is-bluetooth-technology/bluetooth-technology-basics/low-energy

Bluetooth Special Interests Group

Martin Wooly's Blog : http://bluetooth-mdw.blogspot.com/

Dave Smith's Blog :
https://newcircle.com/s/post/1553/bluetooth_smart_le_android_tutorial

Waldner, Jean-Baptiste (2007). Nanoinformatique et intelligence ambiante. Inventer l'Ordinateur du XXIeme Siècle. London: Hermes Science. p. 254. ISBN 2-7462-1516-0

"The Enterprise Internet of Things Market - Business Insider". Business Insider. 25 February 2015. Retrieved 26 June 2015 https://blog.lacklustre.net/posts/BLE_Fun_With_Ubertooth:_Sniffing_Bluetooth_Smart_and_Cracking_Its_Crypto

http://www.kwikset.com/kevo/default.aspx#.ViwM4mvbWVA

http://www.hellokaleido.com/

http://www.cio.com.au/article/439322/pacemaker_hack_can_deliver_deadly_830-volt_jolt/

http://hearinghealthmatters.org/hearinprivatepractice/2014/wireless-pacemakers-need-caution/

http://usenix.org/legacy/event/woot07/tech/full_papers/spill/spill_html/index.html

NOTE OF THANKS :

I would like to thank my supervising professor Dr. Yonghe Liu for his inputs and guidance.

Chunhai Feng for his inputs, help with the testing of the App and feedback on the algorithm.

Dr. Bredow , Dr. Dillon , Madam Pauline , Madam Ann , Madam Janice for attending my defense and helping me out with the administrative aspects of setting the defense up.

Martin Wooley (@bluetooth_mdw) , Mike Ryan, Dave Smith for their inputs and suggestions when I faced roadblocks.

Bluetooth SIG , Google Team for their API , folks at Texas Instruments for their inputs and their resources with documentation.

Krishna Nambiar , Pradeep Mohan, Abhinav Shankar and Joscelyn Sequira for their feedback.