Bluetooth® Security and Privacy Best Practices Guide

- **Revision:** R04
- **Revision Date:** 2023, January 24
- **Prepared By:** Security and Privacy Improvement Program
- **Feedback Email:** security@bluetooth.com

Abstract:

This guide is intended to help implementers better understand why certain available security and privacy choices are better or worse than others for specific applications, and what risks and pitfalls remain in the specifications. It is up to specific implementations to make appropriate choices for their specific security needs that conform to the Bluetooth® specifications. The materials herein are intended to clarify, not specify.

Revision History

Version	Date	Comments
R01	15 April 2020	Initial Version.
R02	10 October 2020	Revised CTKD recommendation. Nomenclature update. Added reflection-attack description. C-8.
R03	08 November 2021	Added recommendation for LE privacy with accept-list. Added description of device-following. Updates for Core Specification 5.3. Added BMS recommendation. Added description of security domain. Added general reflection attack recommendation. Added new CTKD recommendation.
R04	24 January 2023	Added recommendations for limiting the use of RSSI in range measurement. Added recommendations for profile-specific security.

Contributors

Name	Company
Robert Cole	Bluetooth SIG
Andrei Frincu	Bluetooth SIG
Bogdan Alexandru	Bluetooth SIG

Contents

1 Introduction

1.1 Who is this guide for?

Developers, product designers, and technologists who design or deploy systems that use Bluetooth® wireless technology are likely to have at least a passing interest in the security of their implementations. To maintain backward compatibility and to provide maximal flexibility, the Bluetooth specifications allow choices to be made by implementations that affect the security of those implementations, sometimes in negative ways, but always with some justification. Understanding what is possible, what is permitted, and what is recommended is the job of the Bluetooth specifications. Understanding what is “good”, what is “bad”, and when each is true is sometimes addressed by the specifications through its recommendations, but rarely does the Bluetooth specification explain why a choice should be made or explain the consequences of making certain choices.

For those who wish to better understand why certain available choices are better or worse than others for specific applications, and what risks and pitfalls remain in the specifications, this guide is intended to provide a rapid exposure to these details. It is up to specific implementations to make appropriate choices for their specific security needs that conform to the Bluetooth specifications. The materials herein are intended to clarify, not specify.

This guide also attempts to show some of the more subtle ways that errors in implementations may introduce security issues. This additional material represents only a basic overview of the pitfalls; development of secure systems requires vigilance and (ideally) the assistance of automated technologies like static analysis tools. Discussion of these tools and their use is beyond the scope of this guide.

For the purposes of collecting the advice in the best-practices sections into practical groups for different types of implementations, the following logical division is used:

Peripheral – Devices that provide a specific Bluetooth profile, feature, or behavior but do not establish connections to more than one other Bluetooth device. Examples of peripherals include Hands-Free headsets, wireless access points, input devices (mouse, pen, keyboard, scanners), and output devices (printers, displays). The term peripheral may also be used occasionally when referring to the Bluetooth LE Peripheral role (contrast Central role), such uses will be made explicit so there should not be any ambiguity about how the term is being used.

Platform - Devices that connect to one or more Bluetooth peripherals to provide access to the services those peripherals provide. Examples of platforms include workstations, laptops, tablets, and smartphones.

Some end-products do not fall cleanly in either of these two groups. For those types of implementations, how these recommendations apply must be decided on a case-per-case basis.

1.2 Getting the most out of this guide.

This guide lays out some common vectors for malicious attack on Bluetooth devices, many applicable to communications technologies generally. It then enumerates a set of recommendations intended to reduce or eliminate the risk of these attacks in Bluetooth technology specifically by using various Bluetooth features. Some familiarity with these Bluetooth features is assumed in this guide. It is not intended to be a general introduction to the Bluetooth security and privacy features. When unfamiliar jargon is reference in this guide, please refer to the section on Acronyms and Abbreviations (10).

Throughout this guide, direct references to Bluetooth specifications will be made in order to provide detail and show how specific features work. Because the Bluetooth specifications are being updated regularly to make improvements and introduce new features, there is some risk that the specification references in this guide will become obsolete over time. Where the specification revision referenced by this document has been supplanted by a new revision, one or more updates may change what is accurate and complete in this guide. Until this guide is updated to reflect those changes, make certain to follow any advice in new specification language that may supplant or supersede advice provided here. The following specification revisions are referenced in this document (though some references are implicit):

- Bluetooth Core Specification v5.3 (Core)
- Generic Object Exchange Profile 2.1.1 (GOEP)
- Phone Book Access Profile 1.2.3 (PBAP)
- Hands-Free Profile 1.8 (HFP)
- SIM Access Profile 1.1.1 (SAP)
- Bluetooth Network Encapsulation Protocol 1.0 (BNEP)
- Personal Area Networking Profile 1.0 (PAN)
- Blood Pressure Profile 1.1 (BLP)

References to Core Specification language pepper this guide and should be followed to understand the specific details of a feature described in this guide. To avoid constant jumping between footnote numbers and specification references, Core Specification references are included inline within the guide's text. For the Core Specification(s), the following shorthand is used:

[*VOLUME PART, SECTION*]

For example: [6B, 4.6.1] is a reference to "Bluetooth Core Specification Version 5.3 (or later), Volume 6, Part B, Section 4.6.1".

This guide does not delve into an explanation of how to use these referenced features. For a more detailed discussion of the LE security features and practical implementation guidelines, please reference the Bluetooth Low Energy Security Study Guide [1].

1.3 What is Bluetooth Technology?

Bluetooth technology, at its heart, is a suite of wireless communications and protocol standards [2] designed primarily to enable short-range radio communications between devices on the internationally recognized Industrial, Scientific, and Medical (ISM) microwave RF band in the 2.4 to 2.5 gigahertz range. Operating on this specific ISM band permits Bluetooth devices to communicate reliably over short but practical distances without being completely absorbed by common building materials (glass, plexiglass, gypsum, concrete).

1.4 What is Bluetooth Security?

Devices with numerous purposes, power requirements, functional requirements, and security needs can all use the Bluetooth standards to implement their features in various ways to create solutions that will interoperate with other devices that follow those same standards. The Bluetooth standards all have minimum requirements to ensure that the required subset of features will always permit some interoperability. Otherwise, what features a device is required to implement while still conforming to the specification(s) is quite flexible.

Unfortunately, with this flexibility comes risk; it's easy to select security or privacy features for your application that aren't sufficient to secure the confidentiality or integrity of the data that's transmitted or received, or even to neglect to implement those security features entirely. It's also easy to overlook security features that the Bluetooth standard does not currently provide and miss additional layers of access control that may be warranted.

For those readers with a good understanding of Bluetooth technology and the basic Bluetooth security features, it may be useful to skip ahead to the 'Recommendations and Best-Practices' sections. These sections provide guidance on the "best" security features the core Bluetooth technologies offer, and what tradeoffs must be made when choosing those options. In many cases, what security features a Bluetooth device or application choose to use will be dependent on other factors; what's "best" for security may not always be the best choice overall.

The Bluetooth Core Specification defines multiple standards for radio communications between devices; generally portable devices. Broadly speaking, these standards are referred to as:

- **BR/EDR** (**B**asic **R**ate / **E**nhanced **D**ata **R**ate) or Bluetooth Classic
- **LE** (**L**ow **E**nergy) or Bluetooth LE (formerly Bluetooth Smart)
- **AMP** (**A**lternate **M**edium Access Control **P**hysical-Layer). AMP is special in that it's mainly a method for transmitting Bluetooth protocols over other wireless technologies like WiFi. The security features of those other technologies then become relevant but are outside the scope of this guide. AMP is also a legacy feature that is no longer supported in Bluetooth Core Specification version 5.3.

For each of the core standards, there are different techniques, tools and algorithms in use providing the fundamental mechanisms to provide secure transmission of data, and (with LE) to provide mechanisms to thwart attempts to track devices sending LE advertisements.

1.5 What isn't Bluetooth Security?

Many classic Bluetooth profiles act as a cable-replacements for wired connections between devices. Where a Bluetooth protocol standard defines a wireless replacement for a technology, it frequently borrows aspects of the underlying technology. For example, serial-port emulation in Bluetooth technology is through the RFCOMM layer, which is based in part on IrDA (Infrared Data Association) standards, which includes support for classic RS232 signals (e.g. from the DB9 and DB25 serial data cable standards). The data protocols for hands-free devices (the most recognizable public face of Bluetooth technology) are themselves based on a subset of the Hayes modem commands that are still supported by WWAN (cellular telephone) radios. These modem commands are in turn carried over the RFCOMM serial-port emulation layer.

Because the Bluetooth protocols often support the same security domain as protocols designed for use with a wired connection, these classic Bluetooth connections don't intrinsically support extended security measures common on the Internet. They aren't required to support a login and they don't know who you are (or aren't) and whether you're permitted to use a specific profile or take a specific call (for example). When someone tries to connect to your Bluetooth headset without authorization, it isn't required to remember what happened and report it to your phone later (auditing). There isn't a central server which is consulted to see if your headset is still authorized to talk to your phone, rather a local trust agreement exists between these devices: the device pairing/bonding relationship.

Even though LE extends the scope of Bluetooth technology to include unsolicited advertisements of data, it retains a similar scope for security services to the classic Bluetooth standards.

Applications and Bluetooth stack vendors are free to extend Bluetooth services to introduce some of these security features, but there aren't currently standards which define them so that these extended security features can interoperate between different vendor's implementations. Some security features -- for example those requiring communication to a central server-- somewhat defeat the purpose of Bluetooth as a wireless technology that can be used anywhere at any time. Do you want your Bluetooth headset to always require access to a central Internet database before it can establish a call through your phone?

These additional security features are beyond the scope of what the Bluetooth standards currently provide. What the Bluetooth standards do attempt to ensure is that Bluetooth connections are as secure as a wired connection in plain sight would be. It shouldn't be possible to read the information transmitted over the wire, it shouldn't be possible to change the information transmitted over the wire, and it shouldn't be possible to switch the wire out in the middle of an activity (e.g. transferring a file) and get access that way.

LE also adds privacy features. This is where the wired-connection analogy falters. Privacy at the device level isn't trivial to achieve, and BR/EDR, when supported, can thwart some of the privacy measures that LE provides by broadcasting a fixed address during device discovery.

2 Security and Privacy Fundamentals

Security (and privacy as an extension) is generally about one fundamental goal: protecting your information from those who may wish to possess it, change it, or change when you can access it. Why someone wants access to your information is a complex question; what they desire is not always intended to be malicious. Nevertheless, for the purposes of this guide, we're going to treat anyone who wishes to gain unwanted access of any kind to be acting maliciously. Generally, we're going to call them the **'Attacker'** in the following recommendations. To help assist in understanding the basic attacks, the attacker will be also referred to as Eve (alliterative of Eavesdropper) in the following examples and will be attacking communications between hypothetical persons named Alice and Bob.

2.1 Confidentiality, Integrity, and Availability

Attackers who can access information they are not intended to are said to be making the information vulnerable to a loss of **confidentiality**. If Eve can read and comprehend a message that Alice has sent to Bob, then confidentiality has been lost.

Attackers who can modify information they are not intended to are said to be making the information vulnerable to a loss of **integrity**. Integrity means that Bob can be sure that the message received from Alice is the same as the message Alice sent and hasn't been changed by Eve.

Attackers who can modify when you can access information are said to making the information vulnerable to a loss of **availability**. If Eve can stop Bob from receiving the message Alice sent, that is a loss of availability.

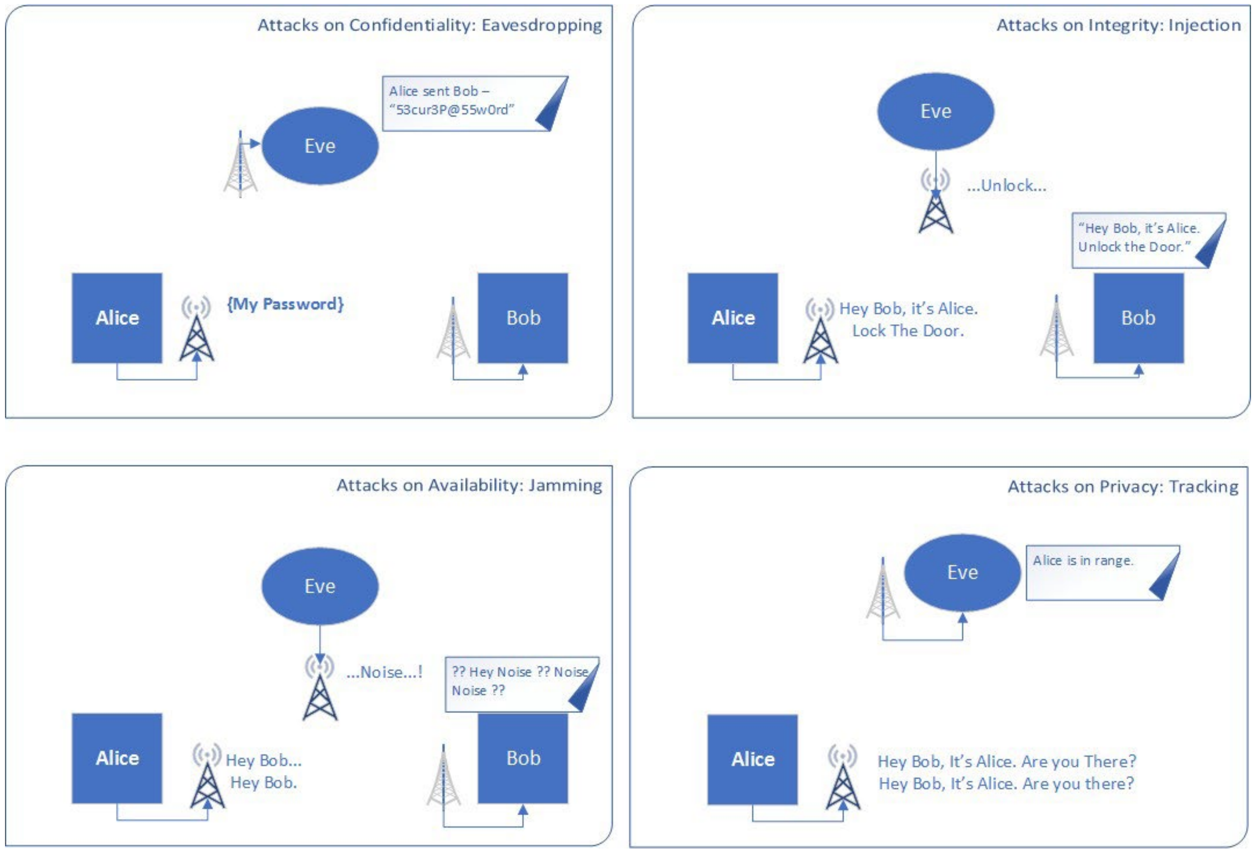
Attacks on Confidentiality: Eavesdropping
Alice sent Bob – "53cur3P@55w0rd"
Eve
Alice
{My Password}
Bob
Attacks on Integrity: Injection
Eve
...Unlock...
"Hey Bob, it's Alice. Unlock the Door."
Alice
Hey Bob, it's Alice. Lock The Door.
Bob
Attacks on Availability: Jamming
Eve
...Noise...!
?? Hey Noise ?? Noise Noise ??
Alice
Hey Bob... Hey Bob.
Bob
Attacks on Privacy: Tracking
Eve
Alice is in range.
Alice
Hey Bob, It's Alice. Are you There? Hey Bob, It's Alice. Are you there?

Figure 1 – Common Attacks

2.2 Authentication, Privacy and Non-Repudiation (other classes of attack)

If Bob receives a message that appears to be from Alice, how does he know whether it was actually sent by Alice or by Eve? If Eve can convince Bob that messages that Eve sends are actually coming from Alice, that is a loss of **authentication.**

If Eve can find out that Alice is talking to Bob, or can find out where Alice is, when Alice or Bob don't want her to know, that is a loss of **privacy**.

If Alice sends a message to Eve and Eve acknowledges receipt, but later Eve denies receiving the message (repudiates it), how could Alice prove to a third-party Bob that Eve did receive that message? The method that Alice uses to prove to Bob receipt of a message by Eve is a form of **non-repudiation**. Proving a non-repudiation to Bob usually requires Bob's involvement when the message is sent, ensuring that Eve is Eve and has acknowledged receipt of Alice's message to Bob. Third-party transactions are not a core part of Bluetooth technology peer-to-peer communications, so this form of non-repudiation is not currently supported by the specifications.

2.3 Impact on Wireless Communications

In the case of wireless communications, you generally want to send and receive data between two devices (or from one to many or many to one), but you want to ensure a number of other things too:

- [Confidentiality]: The only devices that can decode or access your wireless data are those that are intended to.
- [Authentication and Confidentiality]: Any device that receives and can decode your data is the intended recipient. Data you receive can be verified to be from the device you believe sentyou the data.
- [Integrity]: Any data sent is always the same as the data received and is received in itsentirety.
- [Privacy]: The data (or the protocol) doesn't directly identify you as the one who sent the data and any data you receive also does not uniquely identify you.
- [Privacy]: The protocol is intended to make it difficult to track you (e.g. by looking for a unique address that's used for every packet).

2.4 Security Domains

A security domain is a set of resources that a specific actor (e.g. a person, device, or group of people or devices) is permitted to access. A password, key or other shared secret is usually required from the actor to prove its identity to access the security domain. Once the identity is confirmed in a sufficiently non-repudiable way (see Non-Repudiation [2.2]), the resources in the domain become accessible to that actor, though the degree of access may differ for various objects in the domain. Items not accessible are not considered part of the security domain that a specific actor can access, though other otherwise inaccessible domains may contain resources that could be accessible to a different actor. The security domain defines the boundaries between accessible and inaccessible resources.

Bluetooth devices typically offer a single uniform security domain to their peers, though each peer may see a different set of resources, depending on the security levels and encryption key strength used. In this document, security domain generally refers to the profiles, services and specific service characteristics that are accessible to the remote peer. In this sense, the security domain accessible to a peer is often dynamic, increasing in scope as the peer completes encryption, authentication, or authorization procedures.

2.5 Ranging and Range-Based Authorization

One of the techniques for testing whether a wireless peer device is authorized to perform a particular action is to identify its physical distance from the device that is checking that it is authorized to perform that particular action. This is used in devices like key-fobs that are intended to be used only in proximity to a vehicle lock or a door lock or in kiosks to identify when a remote user comes into visual range.

Bluetooth devices do not support range measurements as part of the set of supported features in the Bluetooth Core specification. Range measurements are still possible either through vendor-specific extensions or by relying on a Received Signal Strength Indicator (RSSI) measurement to provide an approximate range measurement. RSSI is provide in decibel milliwatts (dBm), or the number of milliwatts (mW) of power received scaled logarithmically where 0 dBm represents 1 mW, 10 dBm represents 10 mW, 20 dBm is 100 mW, -10 dBM is 0.1 mW, -20 dBm is 0.001 mW and so on.

An RSSI measurement is made available to a host for remote peers in both LE and BR/EDR devices, but BR/EDR power output is adaptive, and a Bluetooth controller normally provides received-power measurements only when they are outside of a specific Golden Receive Power Range. While in this Golden Receive Power Range the value reported is 0 dBm.

In contrast, LE provides a raw dBm value which represents the power received from the remote device. This value is useful only if some method is available to translate received power into a distance measurement. For most device pairs, this would require calibration. When both peers are made to known specifications by a single manufacturer, the calibration can occur at design time or between manufacture and deployment of the device. RSSI still offers only an approximation of range as objects and materials between the receivers can introduce signal attenuation.

In most cases, range-measurement is a useful approximation to identify when a user enters, leaves, or is within a specific radius of a measuring device, but there are methods for manipulating these measurements to make a remote peer seem closer than it actually is; one class of attack is the relay attack (see ‘Relay Attack’ in Common Attacks and Mitigations below).

2.6 Common Attacks and Mitigations

The following section describes some of the common attacks that Bluetooth devices are subject to and some general descriptions of methods for mitigating these attacks. For those already familiar with these topics, it may be useful to proceed directly to the recommendations section.

2.6.1 Eavesdropping

Eavesdropping is when a third-party listens to a conversation without alerting any of the parties directly involved in the conversation to their presence. Bluetooth technology, in common with nearly all consumer radio technologies, is designed to allow devices to communicate with one another omnidirectionally. To permit this, the microwave energy emitted by a Bluetooth device is generally intended to be (approximately) the same at any angle and at any distance from that radio. In part, this means that Bluetooth signals can be received and decoded by anyone within a sufficient range of the antenna to decode signals that are above the noise floor. That means that anyone near a Bluetooth radio will be able to receive every bit of data transmitted between the two devices, permitting undetectable eavesdropping.

A Bluetooth radio changes carrier frequency at regular intervals. For BR/EDR and LE (channel-selection algorithm #2 [6B, 4.5.8.3]), this frequency hopping is done randomly but predictably, based on a shared seed value known by both devices: the Central-role clock in BR/EDR, various event counters in LE. The purpose of this frequency hopping is to reduce interference (both received and generated) but it can also make it more difficult for an eavesdropper with limited resources to begin tracking an already active link.

Hardware is readily available that will track and record the entire available Bluetooth spectrum: wideband sniffers. Using one of these wideband sniffers permits an eavesdropper to record the entire ongoing conversation between two (or more) Bluetooth devices easily. Since an eavesdropper cannot be stopped from recording the conversation, keeping an eavesdropper from listening is not a matter of stopping the acquisition of the transmitted data, rather it is necessary to obfuscate and alter the data in a way so that both the sender and receiver of the data can read it, but a third party cannot. The general technique used by Bluetooth technology to achieve this goal of thwarting an eavesdropper is encryption.

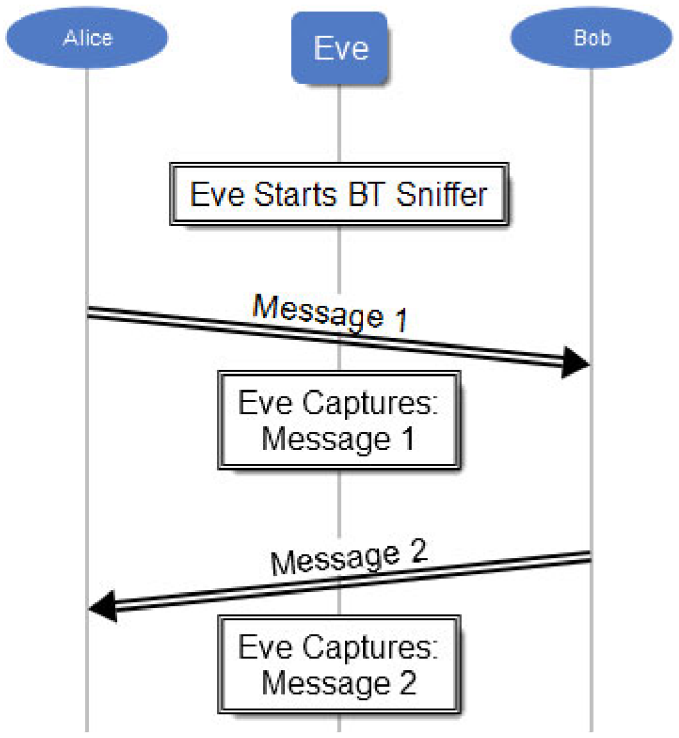

Figure 2 – Eavesdropping with a Sniffer

LE also supports advertisements, which provide a connectionless mechanism for transmitting information. Advertisements are generally intended to be received by multiple LE devices, so eavesdropping advertisements is the intended behavior. For this reason, encrypted advertisements are not supported by the core Bluetooth specification. The Mesh profile specification introduces encryption of data in advertisements when advertised data is used to carry packets between devices in the mesh subnet (the LE advertisement bearer).

Encrypted data is data converted by an algorithm from its original form by a series of permutations, transformations, and substitutions based on a number called a key. The algorithm and the key are used together to produce data that cannot be interpreted by an eavesdropper without access to the correct key, but that can be decrypted, or converted back into the original data by a recipient who possesses either the same key (symmetric encryption) or a secret part of a two-part key (asymmetric encryption). Encrypted data is only as well protected as its key and the difficulty in reversing the action of the algorithm that is used to encrypt the data without the key. Providing a mechanism for securing the exchange of the key when symmetric encryption is used is the purpose of Pairing and Authentication in Bluetooth technology. Pairing (or bonding, if pairing is persistent) is the process of exchanging and/or storing a secret key used to encrypt the link between two devices and identifying those devices to one another. Authentication is the set of mechanisms used during the pairing process to protect against the actions of another type of attacker, the so-called Man-in-the-Middle (MITM).

2.6.2 MITM

Although encryption can use keys to protect the confidentiality of information passed between parties with a shared key, the most efficient encryption algorithms require that both parties share the same key (efficiency being important for battery-powered devices, which Bluetooth devices often are). Part of the problem with initially exchanging information used to subsequently encrypt data is that you don't generally have a mechanism for ensuring that the party that you're getting the key from is actually the party you're trying to send encrypted data to. When encrypting data between two parties Alice and Bob that have not previously communicated, a man-in-the-middle attacker Eve can pretend to be Bob to Alice and pretend to be Alice to Bob, and neither Bob nor Alice will be the wiser. Alice and Bob will each unknowingly encrypt their data to be read by Eve, the **MITM**. Eve can then decrypt, modify and re-encrypt all of the data exchanged between Alice and Bob, relaying messages between them unchanged, or modifying them to suit a possibly nefarious purpose. A successful MITM attack by Eve permits Eve to read all the encrypted data sent between Alice and Bob.

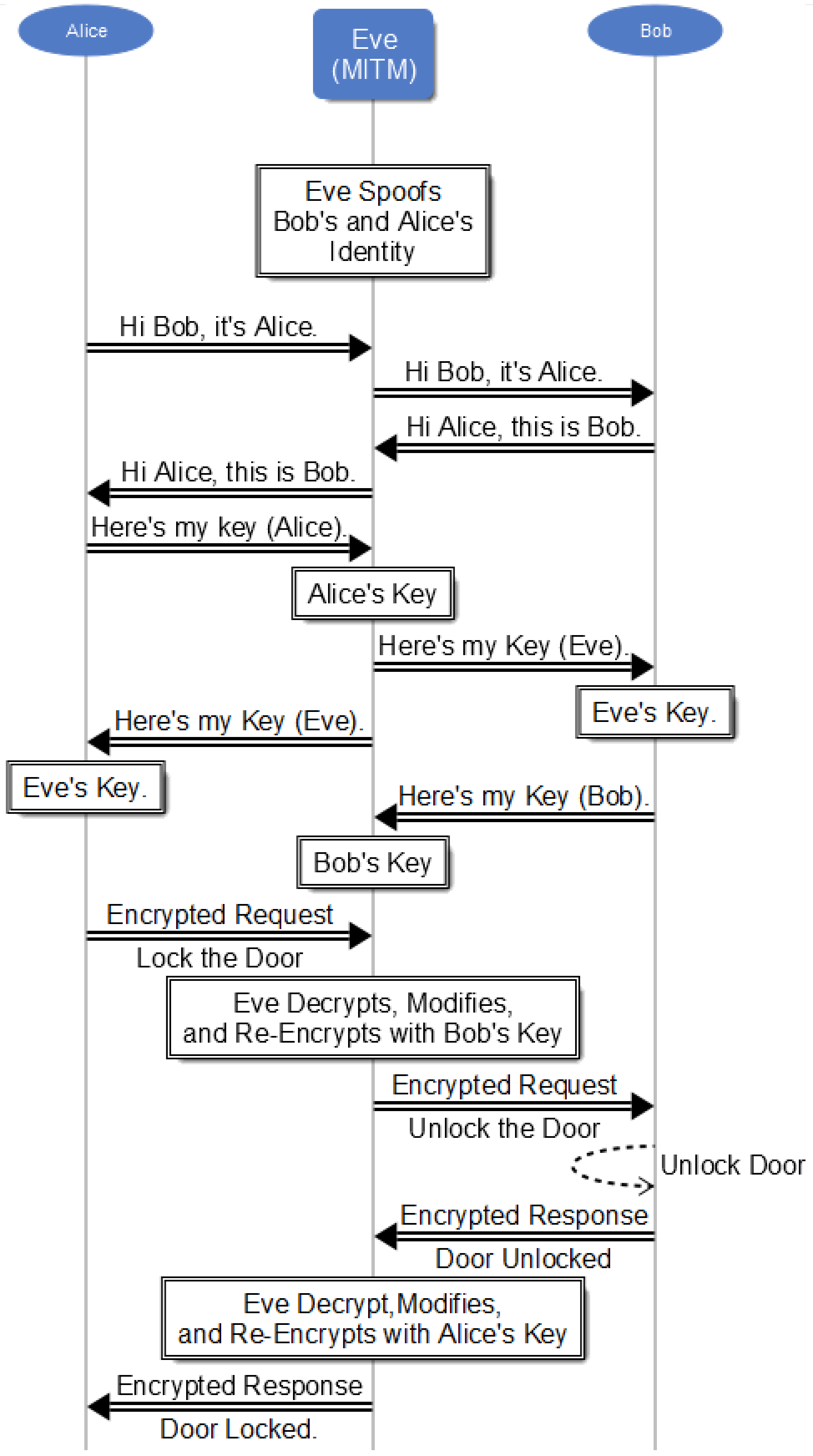

Figure 3 -- MITM Attack

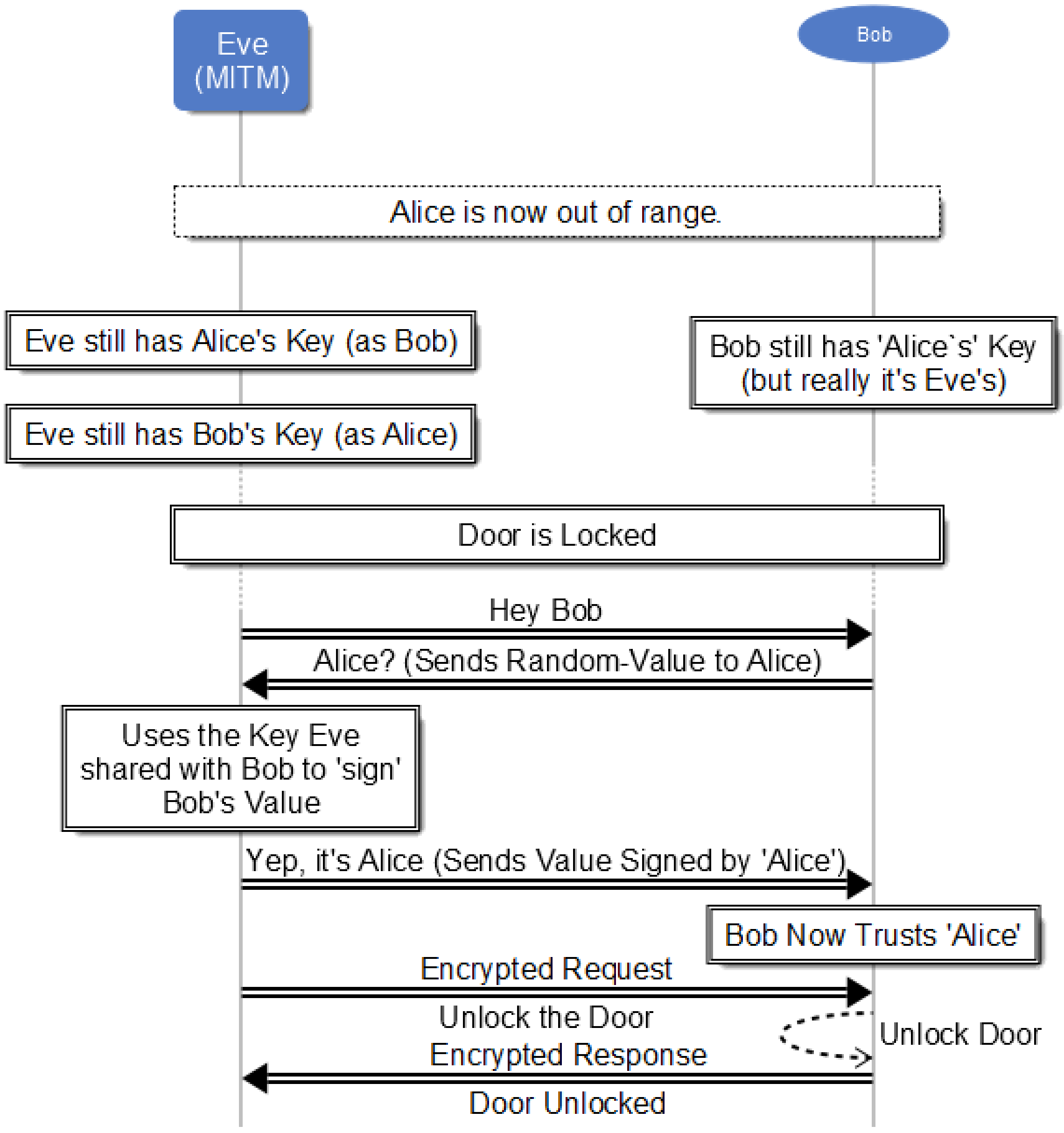
Eve (MITM)
Bob
Alice is now out of range.
Eve still has Alice's Key (as Bob)
Eve still has Bob's Key (as Alice)
Bob still has 'Alice`s' Key (but really it's Eve's)
Door is Locked
Hey Bob
Alice? (Sends Random-Value to Alice)
Uses the Key Eve shared with Bob to 'sign' Bob's Value
Yep, it's Alice (Sends Value Signed by 'Alice')
Bob Now Trusts 'Alice'
Encrypted Request
Unlock the Door
Unlock Door
Encrypted Response
Door Unlocked

Figure 4 -- MITM Attack (spoofing a peer)

Many of the techniques used in a key exchange are designed to thwart the actions of a MITM. All of these techniques boil down to methods of exchanging a secret between the devices that the MITM cannot discover in time to intercede in the key-exchange process. Any attempt to directly exchange a secret via Bluetooth technology can just be intercepted by the MITM, so another mechanism must be used. Bluetooth pairing uses techniques to exchange the secret either external to Bluetooth technology (called Out of Band (OOB)) or by using techniques to share specific numbers that the MITM cannot directly affect without knowledge of two secrets that neither party shares with the other. Though not perfect, these MITM attack mitigation techniques make it very difficult for a MITM to reliably acquire the secret used to encrypt data between devices.

2.6.3 Injection

The packets transmitted by Bluetooth radios can theoretically be spoofed by an attacker by sending an attacked device a message that it believes came from another device with which it is communicating. An **injection** attack is at least plausible with both BR/EDR and LE physical connections.

Packet injection isn't terribly useful with encrypted connections that the attacker cannot decrypt, but there are other control messages that an attacker may be able to spoof which could result in a configuration used between two devices that is more conducive to being exploited than the configuration those devices would select on their own. Feature exchange, key length exchange, and public-key exchange all involve procedures where a spoofed packet may be useful to an attacker.

Injection attacks are usually mitigated through the use of confirmation values and (plausibly) one-way functions (e.g. hashes) to ensure that an injected packet results in a failure of a subsequent step in authentication and encryption. When done properly, this ensures that a device that sends a security parameter has the same value received by the remote party that was sent.

2.6.4 DoS (Interference)

The most successful attacks against radio technologies don't necessarily require breaking the confidentiality or integrity of an encrypted data stream. Instead, making a device unavailable to its user(s) may be enough for an attacker, for example in a control system or a security system. Because a Bluetooth radio can only receive data correctly if it can successfully decode it, signal interference which introduces random noise of sufficient amplitude to all of the channels that Bluetooth technology may use has the ability to prevent the correct reception of any Bluetooth packets. Such radio frequency interference (RFI) can be targeted to just the ISM band where Bluetooth communication occurs, or it can be interference across much of the radio spectrum. In either case, sufficient interference can disrupt Bluetooth communications (as well as other technology using the same ISM band) leading to a denial-of-service (**DoS**).

In contrast to wideband interference, which can disrupt any devices using the frequency ranges being targeted, an attacker aware of the channel hopping sequence in use between two devices can target interference to precisely the frequencies those two devices are transmitting and receiving signals on. This can prevent a specific connection from functioning without affecting other devices. Such targeted attacks require the attacker to have much more information about the attacked device (for example the hopping sequence in use and any masked frequencies not being used).

Bluetooth devices have limited avenues for mitigation of deliberate interference, though the use of AFH [1A, 7.2] to adapt to noisy channels permits normal interference, especially transient interference, to be successfully bypassed in most cases.

2.6.5 DoS (Protocol Attacks)

Denial of service attacks can also take the form of repeated normally valid requests for service by an attacker attempting to block the progression of a connection attempt, or an attacker parking on a connection to block subsequent connection attempts. In such cases, the attacker is not able to acquire confidential information or inject invalid data but is still able to interfere with the proper functioning of the device.

2.6.6 Replay Attacks

A naive encryption implementation might convert every block of data with exactly the same transformation each time it is encrypted. This opens an avenue of attack because that encrypted data, if an attacker has some awareness of the protocol in use, can be used by an attacker at any time to send a packet that it has already seen and therefore appears to be correctly encrypted by its recipient. If the data were known to represent a command that affects the recipient device, the attacker may be able to perform that same action at a later time by simply sending the same encrypted packets again.

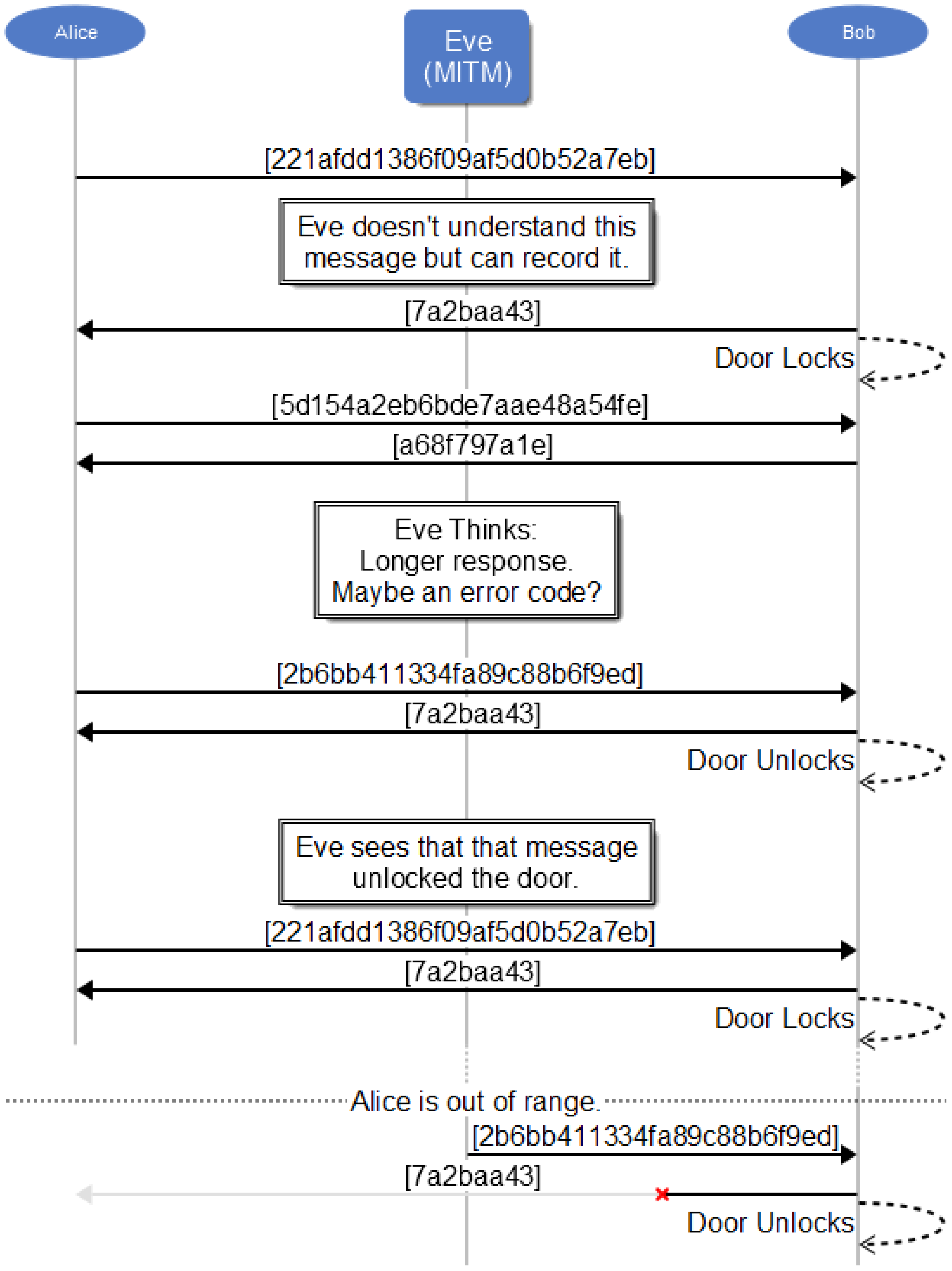
Alice
Eve (MITM)
Bob
[221afdd1386f09af5d0b52a7eb]
Eve doesn't understand this message but can record it.
[7a2baa43]
Door Locks
[5d154a2eb6bde7aae48a54fe]
[a68f797a1e]
Eve Thinks: Longer response. Maybe an error code?
[2b6bb411334fa89c88b6f9ed]
[7a2baa43]
Door Unlocks
Eve sees that that message unlocked the door.
[221afdd1386f09af5d0b52a7eb]
[7a2baa43]
Door Locks
Alice is out of range.
[2b6bb411334fa89c88b6f9ed]
[7a2baa43]
Door Unlocks

Figure 5 -- Replay Attack (Naive Cipher -- ECB)

Even in cases where an encryption algorithm results in a different result each time that the same data is encrypted, it may still be possible for an attacker to block the reception of a packet and use a legitimate (sequential) retry as an opportunity to make the recipient device miss an encrypted packet but be unaware this happened, giving the attacker access to a valid packet that it can play back later.

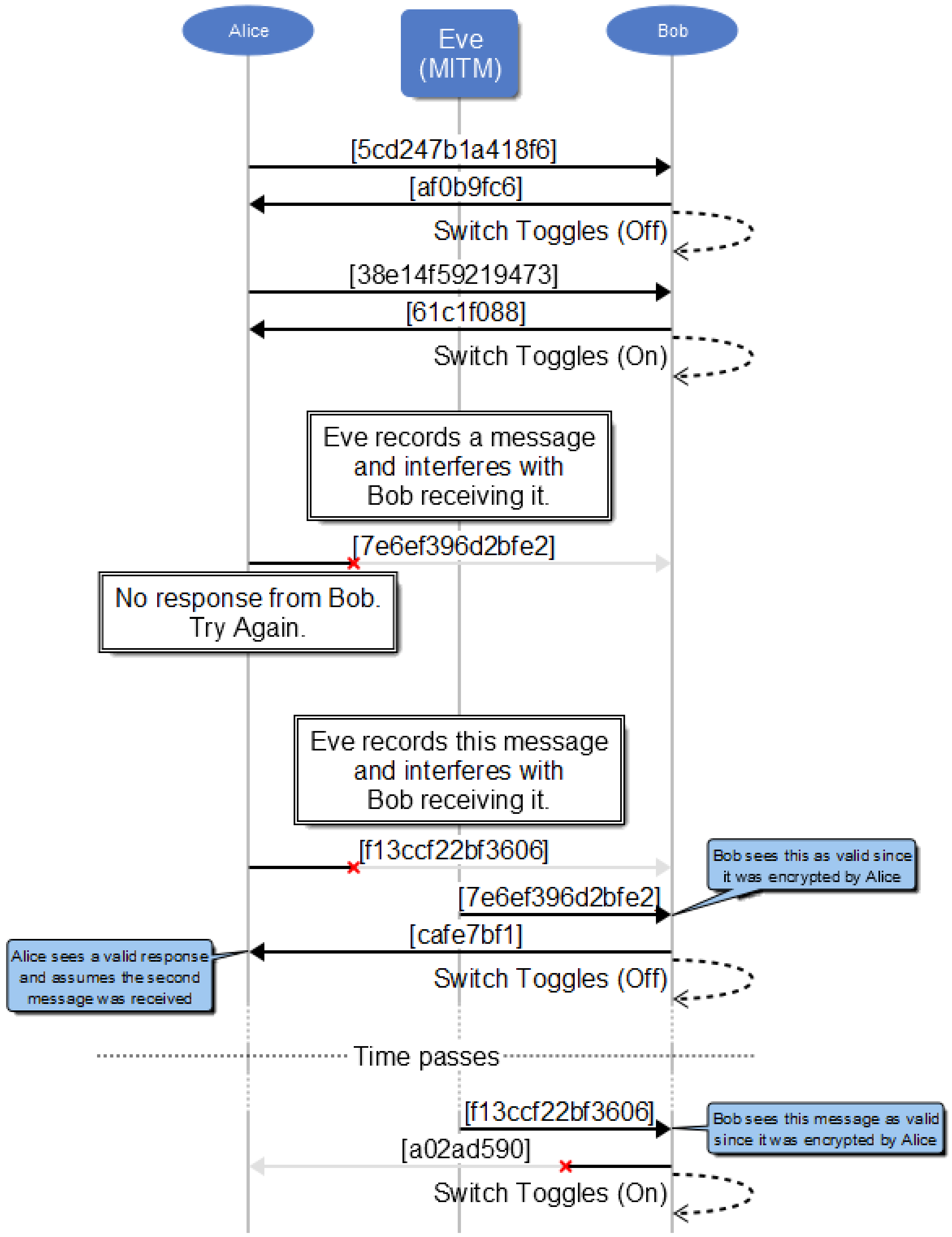
Alice
Eve (MITM)
Bob
[5cd247b1a418f6]
[af0b9fc6]
Switch Toggles (Off)
[38e14f59219473]
[61c1f088]
Switch Toggles (On)
Eve records a message and interferes with Bob receiving it.
[7e6ef396d2bfe2]
No response from Bob. Try Again.
Eve records this message and interferes with Bob receiving it.
[f13ccf22bf3606]
Bob sees this as valid since it was encrypted by Alice
[7e6ef396d2bfe2]
[cafe7bf1]
Alice sees a valid response and assumes the second message was received
Switch Toggles (Off)
Time passes
[f13ccf22bf3606]
Bob sees this message as valid since it was encrypted by Alice
[a02ad590]
Switch Toggles (On)

Figure 6 -- Replay Attack (No Confirmation)

All of the Bluetooth encryption protocols, including legacy BR/EDR encryption (E0), have protections against replay attacks, though at least one available security mechanism ([LE-8] 4.8) doesn't cover all possible replay attack vectors.

2.6.7 Reflection Attacks

A variation on the replay attack is the reflection attack. Authentication procedures and cryptographic algorithms often rely on a party proving possession of a specific secret without revealing the secret. This is usually done by transforming a randomly selected number into another number using a one-way function (or a function that is plausibly one-way, since a general proof that such a function exists is still conjecture). One-way functions (e.g. hash functions) are computationally inexpensive to perform but inverting them is significantly more computationally expensive, and may not be a unique result. The output of the one-way function uses the secret and the random data as input, thus hiding the secret while still allowing someone else with the secret to confirm it was used in the computation, given a copy of the random data. A risk in this approach is that the proof and the random data used to derive it (with the secret) are themselves sufficient to prove possession of the secret, so an attacker can simply reflect the random number and the proof back to make it appear it knows the secret. It's important to not accept authentication evidence identical to evidence previously sent. It's also important to use a new random number (that you produce) each time to ensure that evidence from a previous exchange cannot be reused if the confirmation number (sent first) matches a previous exchange between peers.

2.6.8 Attacks on Secret Keys

The encryption algorithms used by Bluetooth technology are all based upon the techniques of symmetric-key cryptography. A single key is known to all parties that need to encrypt or decrypt packets and must be hidden from anyone else. In the case of the Bluetooth encryption protocols, a different key is used for each encrypted session: the encryption key (or session key). It's derived from the secret shared between peer devices during pairing or bonding (the link key in BR/EDR or the LTK in LE).

Encryption algorithms based on symmetric keys generally use an algorithm that transforms a block of unencrypted data (or plaintext) into an encrypted block (or ciphertext) and then back. The inputs to these encryption algorithms include the plaintext, the secret key, and usually some running counter or feedback from the last encrypted block to ensure that the same plaintext results in a different ciphertext each time it is encrypted anew. Reversing the encryption process must be easy for someone with knowledge of the value of the key and the ciphertext(s), and difficult without it.

2.6.8.1 Brute Force Attacks

An obvious flaw in an encryption algorithm is that anyone correctly guessing the key can completely decode the ciphertext. What if one were to simply try every possible key searching for the one that decrypts the data correctly? The simple answer is that this attack, called brute-forcing, will often work. The limiting factors are how many different key values will result in decrypted data that's meaningful, and how long it takes to try different keys looking for the right one.

If a key as long as the plaintext were used, the resulting ciphertext could still be reversed by trying different keys, but every possible plaintext could be a valid result of deciphering the ciphertext with some value of the key. Trying the only correct key may give back the Bluetooth packets, but another possibly valid key value will give part of the Gettysburg address, or a recipe for cherry pie, or even a different set of Bluetooth packets. Which is the correct interpretation? The attacker may not be able to decide. In this case, the uniqueness of the key is as great as the uniqueness of the data, a property made formal in cryptography by the unicity (or the quality of being unique) of the key against the plaintext. The unicity distance of the cipher is the measure of the amount of ciphertext required before an attacker trying every possible key will be able to reliably select the right key, because only it results in meaningful plaintext.

For the example of a key as long as the plaintext (a form of encryption historically referred to as a one-time pad), the unicity distance is the length of plaintext. Every possible plaintext could be the right one, which limits what an attacker can do with the ciphertext but no key. The problem is that this massive key must be shared before the plaintext is encrypted, it requires as much time to transmit and storage to record as the plaintext, and it can't be shared using the same mechanism that shares the encrypted data, otherwise the attacker will just record it too.

So, the key used is usually much shorter, which drastically lowers the unicity distance. In Bluetooth encryption, the longest supported encryption keys are 128 bits (16 bytes) long. That generally equates to needing only about 20 bytes of ciphertext before a brute-force search for the key will work reliably every time. What stops trying every key from being a viable attack? The number of different keys that need to be checked.

If you pick a key out of a pool of 2^{128}, the chances of randomly picking a matching key are 1 in 340,282,366,920,938,463,463,374,607,431,768,211,456 (approximately $3.4*10^{38}$ or about a third of a duodecillion). Trying them all (or on average ½ of all possibilities) will pretty much ensure success. But there's no way to do that today; the resources are far too limited today to build enough computers and generate enough energy to decode a single Bluetooth link by brute forcing it in a human lifetime.

Is it possible to reverse the encryption process without knowing the key? That depends on the algorithm. A good encryption algorithm makes each bit of the ciphertext dependent on the key and plaintext in ways that resists reversal. Values for a trial key with only a single bit different from the real key will generally decode to plaintext gibberish (the avalanche effect **[**17**]).** Could a future mathematical breakthrough make the reversal of a specific encryption algorithm (e.g. AES) without the key trivially easy? Possibly, but it's not a good bet.

2.6.8.2 Keys and Entropy

An important caveat in selecting an encryption key is to always build the key from bits that are truly random. If an attacker can find the same key that you selected by knowing details about your starting conditions (e.g. with insecure random number generation) or even identifying some of bits in the key you selected, the key may no longer be difficult to brute force. Achieving this randomness requires generating keys using a source of bits with high information entropy (or just entropy). **Entropy** is a measure of how unlikely a specific data value is from the pool of possible values it was sourced from. A fair coin flip (ignoring the edge) has two outcomes, so each is equally likely (or equally surprising). A coin flip therefore has an entropy of 1 bit; it is just as surprising to get heads as it is to get tails. A perfect key generator would use a source of bits with as many bits of entropy as are in the key, ensuring that one key is no more likely than any other. In practice, for 128 bits, there is no available data source with 2^{128} total possible unique outcomes that are all equally likely to occur. Instead, the bits selected for a key must be built up from multiple measurements or derived via an algorithm that presents a good approximation to a real source of entropy. Good sources of entropy include real-world random processes measured repeatedly over time (e.g. coin flips, thermal noise, radionuclide decay). Poor sources introduce correlations which decrease how surprising (unlikely) a specific outcome is. People, for example, are quite bad at choosing random numbers.

It's important to note that information entropy doesn't increase if a key is transformed by a predictable algorithm (e.g. a hash function). A number that looks random is only as random as the entropy of its source.

2.6.8.3 Key Exchange: Sharing the Secret Key

As long as the attacker doesn't have access to the shared symmetric key, encrypted data can only be effectively attacked by brute-force methods. But two devices with a symmetric key must have agreed upon that key to start encrypting/decrypting data, and the act of sharing the key immediately risks it being captured by an eavesdropper while it's being shared.

The most secure approach that Bluetooth technology provides to circumvent this problem is a Diffie-Hellman key exchange [15], a method for agreeing on a number (the key) without either party actually revealing the number directly to the other. Legacy authentication and encryption in BR/EDR and LE use other mechanisms without some of the advantages of the Diffie-Hellman algorithm (i.e. difficulty in brute forcing the key using the information in the key exchange).

The Bluetooth variant of the Diffie-Hellman key exchange algorithm uses a product operation (elliptic curve scalar multiplication) on a finite field over an elliptic curve (either NIST P-192 or P-256 [3]) to share a discrete multiple of a publicly shared point on the curve as the Diffie-Hellman key while concealing which multiple the point represents. This multiple is made difficult to discover by Eve due to the properties of the discrete logarithm problem over elliptic curve scalar multiplication [16]. When Alice multiplies her secret by Bob's public Diffie-Hellman key and Bob multiplies his secret by Alice's public Diffie-Hellman key, they each come up with the same result, because each of Alice's and Bob's secrets are a divisor of their respective Diffie-Hellman key: both Alice and Bob get the same product. But Eve doesn't know either secret, so she doesn't know what that final product is. The secret key (the product), is therefore protected.

As long as the Diffie-Hellman key exchange is approximately as difficult to brute force as the key itself, eavesdropping the key exchange does not introduce additional risk to Alice or Bob.

In Bluetooth protocols, the key exchange occurs only during the pairing or bonding process. In theory, bonded devices do not need to repeat this process, so an attacker cannot acquire the keys through the authentication process after the pairing process has completed. In practice triggering devices to pair again is usually not difficult, but for platforms it rarely works without some notification to the user.

2.6.8.4 Brute Forcing a Key Exchange and Forward Secrecy

If an eavesdropper records the Elliptic Curve Diffie Hellman (ECDH) key exchange, recovering the key is possible if the secret (point multiple) can be identified, but doing so requires brute forcing a discrete logarithm problem on a large finite cyclic group. When properties of the elliptic curve and the public and private key are chosen carefully (e.g. of prime order), brute forcing the ECDH private key is difficult. It's comparable to brute forcing the key directly given only a recording of the encrypted link.

This is true for a single ECDH key exchange, but repeated pairing with the same elliptic curve key pair can provide hints to an attacker or permit the attacker to reuse part of an exchange observed previously. Eve may repeatedly attempt and fail the pairing process just to get Alice or Bob to keep generating ECDH keys for specific random numbers (that Eve picks carefully). If the algorithms which compute the points are not selected carefully, this may provide clues as to which private key is used using side-channel attacks [18]**.**

The trick to avoiding this gradual leak of information to ease brute forcing key exchange is to switch to new points on the curve. This takes away the a-priori information Eve has amassed and makes the next pairing difficult to brute force again. This switching of points on the elliptic curve has the added advantage of making an attack on one pairing session unable to affect the security of a different pairing session using different underlying asymmetric keys. There's no relationship between the pairing and authentication procedures for each new asymmetric key pair, and each is as difficult to brute-force as the next. This is **forward secrecy**. Compromising asymmetric keys for a prior pairing/bonding will not affect a future pairing using different asymmetric keys.

Achieving forward secrecy requires that the device periodically generate new keys, which consumes time and energy. For some devices, especially small battery-powered LE devices, refreshing the key frequently may be inadvisable or may be used as a denial-of-service attack by deliberately draining a device's battery.

2.6.9 Trash-Can Attacks

In some cases, the credentials that a device stores may provide more access than a peer to peer connection. In cases where a device is part of a group of nodes (e.g. a mesh subnet) or a group of peers (e.g. broadcast encryption), if the device is decommissioned without erasing, invalidating or refreshing any credentials the device stores, an attacker may literally acquire the device from the trash (or recycling center) to gain access to those credentials. In most cases, this can be avoided by erasing a device (e.g. by returning to factory settings) before permanently decommissioning it. In cases where a device failure prevents a proper erasure of stored credentials, it is important to dispose of the device properly or update the credentials of other devices in the group so that the disposed device no longer possesses valid keys.

A subset of the trash-can attack is the theft of a device for the purposes of acquiring stored credentials and using them to access other devices in a group.

2.6.10 Relay Attacks

Relay attacks (not to be confused with the similar sounding replay attacks) are attacks where a MITM attacker relays the signals received from one peer device to another peer device at a distance, generally modifying them in the process, for example by amplifying, delaying, or shifting their phase. A relay attack is a class of MITM attack that can be used by an attacker to manipulate the RSSI of a signal to make a device measure its peer as being closer than it really is. When the distance between peers is used as a condition for performing some action, for example triggering a remote-keyless-entry, the attacker may be able to make it appear as if devices are in proximity when they are potentially at a significant distance or behind some physical barrier (e.g. inside a building). This may allow an attacker to gain access to a vehicle, a door or a facility by spoofing the relative location of a wireless peer (e.g. a key-fob) without needing to directly manipulate any of the data transmitted between the peers.

Mitigations for relay attacks exist, but generally require a second source of position information or more sophisticated methods than are available using just measurements of received signal strength. Attackers may also manipulate the received signal to appear to arrive earlier than it normally would at that distance or manipulate the phase offset of the signal to a similar end. These variations on the relay attack affect other methods for confirming range, and each require their own type of mitigation.

3 Common Bluetooth® Implementation Best Practices

3.1 [C-1] Follow the guidance of the latest version of the Bluetooth® specifications.

Why should I do this?

The Bluetooth specifications define what Bluetooth technology is and how it works. Conformant implementations must always follow current specifications but may also implement to older but still active specification revisions if necessitated by the controller or host stack limitations. Newer versions of implementations, in addition to introducing new features, improve security-related features or eliminate vulnerabilities discovered in older versions of the specifications. To ensure that the best security features are supported, use the latest version and implement the best supported security features available. This does not always preclude support for backward compatibility, though in some cases using the latest security features implies deprecating or eliminating support for features that were found to be insecure (e.g. by selecting Secure-Connections-Only Mode). In cases where an optional security or privacy-related feature or configuration is recommended by the specification, that feature should be provided or that configuration selected whenever possible.

Why might I not do this?

In situations where a hardware platform meeting the requirements of the latest specification is not available or does not meet the requirements of an implementation, it may be necessary to use an older version of a Bluetooth specification in a design. An implementation selecting an older but not yet deprecated design risks introducing known vulnerabilities. The choice of whether a security or privacy feature is required and whether that overrides any rationale for choosing a part conforming to an older version of the specification must be made on a case per case basis.

3.2 [C-2] Assess and document the security requirements of any design to ensure appropriate security is used in an implementation.

Why should I do this?

Understanding what type of data is managed by the profiles of an implementation, what risks to user privacy might exist for a specific type of device (especially wearable devices), and what risks exist if the confidentiality or integrity of data carried by a Bluetooth profile are compromised are essential when assessing the appropriate security features to use when designing any product. They're just as important when designing a Bluetooth implementation, because of the relative ease of eavesdropping any wireless technology. Assessing the security needs of an implementation will help to define what security features are appropriate. Documenting these needs can help in selecting the appropriate minimum set of security features based on the effects of certain attacks on a Bluetooth device. For example, a pair of portable headphones may appear to need limited security features, but if such headphones are used to carry a phone call, the confidentiality of the audio data may be essential: encryption should be used and MITM protection should be supported. A set of portable headphones may appear to need limited privacy features, but it's important that such a device not be readily tracked. Whether this privacy is achieved through the use of random addresses, or by simply not broadcasting any identifying information while the device is paired and connected will depend on the use cases. Determining such use cases is the first step in assessing and documenting security requirements.

Although not exhaustive, here are some of the security and privacy considerations that may be important for an implementation to consider:

- Is the device portable? If it's portable, under what circumstances can a user be tracked by someone with awareness of the device's static or public address? For how long can this tracking proceed before the device either becomes invisible to an eavesdropper or changes addresses? When the device changes addresses, is there another method for following the device (e.g. through a unique payload in an LE advertisement)?
- What kinds of static user data are accessible via the device's Bluetooth profiles? What are the risks to the user of the device if an attacker could gain unrestricted access to the Bluetooth interface?
- Does a device provide access to a physical resource of value to its user (e.g. a safe or a door lock)?
- Does a device provide access to medical or other personal data that is restricted under records privacy acts in various jurisdictions? Are steps beyond what Bluetooth device security can provide necessary to protect this information?
- What data is normally carried by the device profiles? What risks exist to a user if this data is compromised? For example:
  - A keyboard may accept the entry of user passwords or pin codes.
  - A fitness tracker may transmit personal medical details.
  - A headset may carry a voice call including confidential (banking/medical/personal) details.

Answers to these and other similar questions should drive what Bluetooth security and privacy features and other security and privacy features are desirable or required in an implementation.

Why might I not do this?

Even if you are implementing a turnkey solution, it is important to still identify the ways that information carried by Bluetooth profiles might be compromised and the importance of appropriate security features to identify whether the provided solution meets the needs of the users of a device.

3.3 [C-3] Ensure that the minimum security requirements of any supported profiles (including custom profiles) are met.

Why should I do this?

Certain Bluetooth profiles identify specific requirements related to the security features and configuration required when using the profile. For example, the classic SAP profile required a minimum length of encryption key for any channel carrying SIM data. In cases where a profile either recommends or requires a specific security or privacy feature be used, this is usually because the data carried by or the features accessed by the profile represent a specific risk to an end-user if compromised by one of the common attacks. Unless compelling reasons exist for bypassing or reducing these requirements, any recommendations for minimum security features should be followed. In contrast, required minimums specified by profiles are not optional and must be supported for a profile to conform to its specification.

Although custom profiles don't specify minimum security requirements, if a custom profile implements similar features to a profile that does recommend or require minimum security features, a similar set of recommendations should be adopted for the custom profile to ensure that similar security is maintained. For example, a custom health profile should adopt at least the same encryption key length and GATT security attributes as other health profiles with similar purposes.

This does not preclude the use of better security for profiles with minimum requirements, though it is important to understand in what ways a specific feature or setting may be 'better' for a specific purpose (e.g. thwarting a MITM attack) before selecting it.

Why might I not do this?

For a profile that recommends some minimum security requirements (e.g. using authenticated GATT characteristics) but does not require them, an implementation is permitted to choose not to implement a security recommendation. In cases where a profile is being used in an unorthodox manner that does not expose risk in the same manner as its normal use (e.g. a testing platform), it may be appropriate to ignore a security recommendation. This sort of scenario is rare. In general, devices should follow recommendations to use security and privacy features.

3.4 [C-4] Test and audit the security features of implementations.

Why should I do this?

The current qualification tools that the Bluetooth SIG provides members test the behavior of most required security features of components or end-products, but they do not validate whether recommendations are followed, they do not test each LE attribute, and they can't make a judgment about the security needs of a specific use of a product or determine what security vulnerabilities may exist in a custom profile with custom security features.

It is valuable for implementations to independently test their security features and audit implementations with independent security experts or using more sophisticated tools (e.g. penetration testing and fuzzing tools). The degree to which additional testing and auditing are valuable is dependent on the scope of a device, the breadth of features it implements, and who will use it.

Why might I not do this?

Even if a product is a combination of existing components and profiles or a rebranding of an existing product, there is still value in at least validating the security features of an implementation. If an obvious security hole exists in an implementation that has been re-branded, the potential damage to the brand due to exploitation of that security hole suggests there still may be value in independent testing of the security features of the device. A decision needs to be made with any implementation whether the cost of having security vulnerabilities outweighs the cost of finding them before release.

3.5 [C-5] Ensure that a UX/UI provides appropriate notification to users of any security or privacy issues.

Why should I do this?

Informing a Bluetooth user that a specific security or privacy risk exists and providing possible mitigations for such risk can be beneficial to reduce security or privacy risks. The Bluetooth specifications neither recommend nor explicitly restrict most user interface design choices. This provides quite a bit of flexibility for user experience design where a platform or peripheral provides a rich user experience. Issuing a notification or warning when a remote peripheral does not provide the latest security features or does not implement device privacy can be beneficial to end-users with specific security or privacy requirements.

Why might I not do this?

In cases where a security issue with a peripheral device could expose the user to risk, identifying that the risk exists doesn't necessarily provide a method for reducing that risk for the user apart from selecting (i.e. purchasing) a newer peripheral (or upgrading firmware, if applicable). When the user is provided with a notification that a device is potentially insecure, but this notification does not provide a mitigation, the ability of the end-user to make an informed choice about the issue may be limited. Where a specific technology exposes severe risk (e.g. HID with legacy BR/EDR pin-code pairing), a warning may be justified. In other cases, the marginal improvement in security afforded by indicating that a specific device does not support the latest pairing procedure only manifests if this prompts the user to forgo use of said device.

3.6 [C-6] Enforce secure coding practices in the development of any interface facing external data sources, especially wireless ones.

Wireless interfaces expose the consumer of data received over the air to any attacker in wireless range capable of transmitting packets sufficiently conforming to the wireless standard to be accepted and passed to a higher layer. At each layer from the radio receiver to the user interface, a malformed or crafted packet received via a wireless radio exposes interfaces to failures using a variety of techniques, for example:

- Buffer Overflow: Packet data must be stored locally, at least temporarily, and if the boundaries of the data are defined by parameters within the packet or are not strictly bounded when moved or copied in memory, it is possible for a crafted packet to trigger a write outside the buffers set aside to store it.
- Code Injection: Where a buffer overflow results in an overwrite of the stack (or other executable memory) it becomes possible for the program counter (or equivalent) to return to an address within the stack containing data from an attacker. This may permit an attacker to jump to arbitrary code, or in platforms without protection against execution from stack memory, for code in the packet data to be run directly from the stack. This may permit an attacker to gain undesirable access, or in extreme cases to even to take control of the platform.
- Infinite Loop: Where a code path exists through a loop structure (or other state machine) with some exit condition or state dependent on data received from a remote device, crafted payloads may cause the loop to never exit (for example by providing an incomplete payload or an invalid data type). Without watchdog or similar computer-operating-properly (COP) circuitry, an embedded processor may become locked in this loop and unable to service other tasks (deadlocked).

These are a small subset of many different attacks a remote attacker may launch. For more information on secure coding practices (for a C/C++ audience) see [19] [20].

Why should I do this?

In order to ensure stability and correct function of a program, invalid inputs to the program must not result in unexpected or invalid behavior. Information from a remote device, even if provided by a party with a trust relationship with the local device, should not be assumed correct without validation. Packet lengths, parameter values, data formats, and procedure-order should be assumed invalid until explicitly validated. Without protection against common exploits, a remote attacker may be able to cause undesirable behavior, including accessing protected memory (e.g. keys, passwords or private user information), taking control of a device or the hardware a device supports, or even just denying an authorized user access to the device.

Why might I not do this?

Always do this. Some form of static analysis tool is offered for nearly all common programming languages, even tools that are even freely available or built into commercial compiler toolchains. Although no tool can find all possible faults (e.g. in light of the halting problem [21]), even the most rudimentary analysis tools can find some errors that could result in remote exploits. For statically-typed compiled languages, code should always be written securely and tested with automated tools to find errors. For dynamically-typed and interpreted languages, identifying failing code paths is often more difficult, but defensive programming techniques can help to reduce the total number of cases that need to be explicitly tested. Using a language (or language environment) with some built-in protections (e.g. Rust, Java, CIL) may also help avoid common secure-coding issues when such languages can be used to develop some or all of an implementation.

3.7 [C-7] For both LE and BR/EDR authentication procedures, limit repeated failed attempts if the host stack does not provide acceptable default limits.

The specification [2H, 5.1 and 3H, 2.3.6] requires that an implementation provide an exponential back-off time for repeated failed authentication attempts. This introduces a waiting interval between authentication attempts from the same remote address, which reduces the ability of an attacker to repeatedly try authentication or to use crafted payloads to incrementally gain information about the underlying keys used.

Why should I do this?

If a Bluetooth host stack implementation provides full support for authentication and encryption behaviors, it is likely that the repeated-attempts back off behavior is incorporated into the host stack. If such a feature is configurable by an implementation or must be introduced at the application layer by an implementation, it is valuable that this feature be enabled (or introduced), otherwise an attacker can rapidly attempt multiple authentications, either to brute-force authentication or potentially to reduce the number of attempts required by using crafted ‘random’ numbers to attempt to reduce the number of possible outcomes in the hashes used in the authentication algorithms (especially for legacy BR/EDR pairing modes). Slowing the rate at which new attempts can be made reduces or eliminates the ability of an attacker to brute-force authentication by any method.

Why might I not do this?

The specification requires that an implementation do this, however the duration of the waiting interval used and the maximum waiting interval used may not be configured initially and can require that defaults be provided to a host stack by the application layer. In other cases, the defaults may be sufficient and an implementation may not need to provide additional protections against repeated failed pairing attempts. Otherwise, an implementation must implement this protection to be conformant to the Bluetooth specification.

3.8 [C-8] Do not accept public keys or authentication evidence that you produce.

Why should I do this?

A device that produces authentication evidence based on a random value may be fooled into accepting forged evidence if it simply accepts its own evidence from another party. This is similar to accepting a password from someone to whom you just revealed the password. It depends on the algorithm how this case may manifest, but in general, authentication evidence that you present to a third party must not be treated as valid evidence if it is reflected back to you.

Similarly, a device revealing the same public key as your public key will not know the private key and cannot use the public key to complete a Diffie Hellman key exchange, but if the public keys are used in an authentication algorithm as inputs, as they are in certain Bluetooth secure authentication procedures, using the same two public keys can result in attacks on authentication.

Why might I not do this?

Always do this. If the random number generated in the authentication process or the public key generated happen to match the value used by the remote peer, it is necessary to fail the security procedure and start again. There are authentication methods where the peer evidence will not match for the same input; for such procedures it may be valid to receive a duplicate value for part of the authentication evidence. In general, however, and certainly with the current suite of authentication and encryption procedures in Bluetooth BR/EDR and Bluetooth LE, duplicate responses are only a method for an attacker to misrepresent their status and should be rejected.

4 Bluetooth® LE Recommendations and Best Practices

4.1 [LE-1] Use LE Security Mode 1, Level 4 to Authenticate and Encrypt Physical Connections.

LE Security Mode 1 offers four security levels. Level 4 requires an authenticated link, exchanges the resulting symmetric key using Elliptic Curve Diffie-Hellman key exchange using the NIST P-256 curve [3], and uses AES-CCM [6E, 1] with the shared 128-bit symmetric key (LTK). It is the most secure option available for LE connections.

Level 4 also requires that a 16-octet encryption key be used. This ensures that the encrypted link cannot be reasonably decrypted by brute-force methods (i.e. trying all permitted combinations for 16-octet keys requires on average 170,141,183,460,469,231,731,687,303,715,884,105,728 attempts).

Why should I do this?

In addition to sharing an LTK, LE Security Mode 1 Level 4 requires the value of a secret number to be confirmed between two pairing devices either by an Out-of-Band (non Bluetooth) communications mechanism, through a visual confirmation of matching randomly generated numbers by a human intermediary, or through entry of a fixed or randomly generated passkey. Each of the supported authentication mechanisms forces a secret number to be shared that a third-party (MITM) cannot directly observe because it is either not exchanged via Bluetooth technology or exchanged over Bluetooth technology using a mechanism that obscures the secret from direct observation see (introduction to the Diffie Hellman key exchange).

Once the LTK has been shared between the two paired devices, access by an attacker to the secret shared during the authentication procedure will not affect the secrecy of the LTK. This is true for all LE Secure Connections authentication procedures. This property of forward secrecy limits the scope of an attack on a pairing procedure to the time during which the pairing procedure is happening.

What are the implications of doing this?

Using Security Mode 1 Level 4 ensures that a MITM attacker will fail to intercede in the key exchange procedure with a probability of 99.9999% or greater (a 1/1000000 chance of randomly picking the correct numeric-comparison value). It ensures that if an attacker records the pairing procedure they will not then be able to acquire the LTK shared during the exchange by discovering the value of the shared secret (not the LTK itself). It ensures that an attacker with the complete recording of the transmissions between the devices will require at most 2^128 trials (on average 2^127) to search for the key, a process estimated to take longer than the current age of the observable universe using available equipment. Together these limitations provide good insurance against attacks on the integrity or confidentiality of the data transmitted and received.

When should I do this?

If any profiles your device supports transmit data that should not be readable by a third-party, or if any profiles your device support accept input that must not be manipulated by a third-party, encryption is required, and pairing or bonding must be used to enable encryption. For any of these cases, the most secure mechanism that can be supported by your device must always be used. Therefore, unless there exists an insurmountable impediment to using security mode 1 level 4 in a new design, it should always be both supported and used.

Why might I not do this?

An older (pre 4.2) device might not support LE Secure Connections. Pairing with that device may require falling back on a LE Legacy Pairing mechanism. An implementation may restrict pairing to only those devices that support LE Secure Connections, but this is not always practical when backward compatibility is required.

Level 4 authenticated pairing may not be possible if one (or more) of the following is true:

- A device has no support for an OOB communications mechanism.
- A device has no display that could support a visual or auditory representation of a six-digit number.
- A device with a display but no sound-source (or equivalent) requires accessibility for the blind.
- A device with a sound-source but no display (or equivalent) requires accessibility for the deaf or hearing-impaired.
- A device has no input that could support a yes/no response (indicating whether numbers match or do not).
- A device has no method of entering a six-digit number or is paired with a device that cannot represent such a number to be entered.

It is recommended that device manufacturers include support either for OOB [3H, 2.3.5.6.4] or for exchanging the pairing secret via the Numeric Compare [3H, 2.3.5.6.2] or Passkey Entry [3H, 2.3.5.6.3] procedures. These are the available mechanisms for thwarting a MITM from potentially interceding in the pairing process and gaining access to the LTK that secures the link between the attacked devices (aside from not being in radio range of a potential attacker).

4.2 [LE-2] Require authentication and encryption for modifiable GATT characteristics.

LE GATT 3G [10] characteristics all have associated authentication, encryption and authorization flags ([3F, 3.2.5] and [3F, 4]) that can be enabled individually. Except in very specific cases, the security requirements for characteristics used in profiles, even standard profiles, are up to the individual implementer's discretion.

Requiring authorization for certain characteristics is also potentially valuable and should be considered as an alternative or in addition to this recommendation (**[LE-18] 4.18**).

Why should I do this?

Access to writable GATT characteristics without authentication and encryption enabled risks a MITM or other attacker being able to either write the value of the attribute without intervention of the authenticated part or modify an ongoing write to change the value of an attribute. In cases where the attribute controls some mission-critical features of an application (e.g. a locking mechanism), it's essential that an unauthorized party not be able to manipulate the value. For less critical applications, it's still rarely desirable for the attribute to be written by an unauthorized party.

What are the implications of doing this?

Requiring authentication for accessing an attribute requires that an authenticated link be used. This means that JustWorks pairing methods will not be sufficient in order to secure access to the value of that attribute. The devices involved must be able to authenticate the link in order to be able to support authentication on GATT characteristics and their attribute values.

Requiring encryption for accessing an attribute requires that an encrypted link be used. Any supported pairing technique may be used to share the keys required in order to enable encryption between the device with the GATT server and its client.

When should I do this?

Whenever a device can support authentication, and for all devices that support encryption, enabling the authentication-required or encryption-required permissions provides protection for any writable attributes against an attacker modifying the value of an attribute. This is essential whenever access to the device is normally restricted (e.g. is part of a security system or is in the possession of a user). Some characteristics, even writable characteristics, don't necessarily require encryption or authentication, but those cases should normally be rare.

Why might I not do this?

In those few cases where access to a writable attribute does not impact the behavior of a device that would affect a user appreciably, it may be valid to not require encryption. In those cases where a device may need to be used without access to a mechanism that supports authorization (i.e. only JustWorks pairing will work), it may be necessary to not require authentication. In most cases, there exists some mechanism by which an attacker may adversely affect the user of a device through a writable attribute; something as simple as consuming power by leaving the device enabled.

4.3 [LE-3] Design Devices that Support Authenticated Pairing using an Input, Output or OOB Mechanism.

Authenticated LE encryption requires the exchange of a secret between pairing devices that the MITM attacker cannot directly observe. For the purposes of Bluetooth pairing, it is generally assumed that a MITM attacker is present within the radio range of the pairing devices and can record Bluetooth radio traffic. The best means of ensuring a MITM cannot acquire the secret shared during pairing is to require the exchange of the secret using an Out-of-Band (OOB) mechanism that only works when devices are in close proximity or require specific line-of-sight access. If such an OOB mechanism is not supported, devices using LE should be designed to support one of the other authentication mechanisms [3H, 2.3.5] using an input or output method.

Why should I do this?

Providing a method for exchanging a secret via an OOB mechanism or for confirming that a secret matches between devices can stop a MITM attacker from intervening in a pairing exchange and pairing with device A as device B and device B as device A, compromising the integrity and confidentiality of the encrypted link while the MITM is present [3H, 2.3.1].

For profiles that carry sensitive data that is not protected from eavesdropping or modification by non-Bluetooth mechanisms, preventing a MITM from decrypting or manipulating the data exchanged between paired devices is essential. The only means to guarantee this protection is to use authentication.

What are the implications of doing this?

Without MITM protection, an attacker may intercede in the pairing process by pretending to be each of the pairing parties to the other. This permits the MITM to exchange an LTK with each party and have the ability to decrypt all data exchanged and modified data as the attacker sees fit. Practically, launching such a MITM attack is not trivial, as the MITM must spoof the identity of the attacked devices and get both to connect to the MITM. Additionally, the MITM can only decrypt the link while physically adjacent to the attacked devices. Once the two devices are moved away from the MITM's radio range, they will no longer appear to be paired with one another, and if they pair again without the MITM present, further intercession by the MITM will be thwarted.

When should I do this?

Unless hardware limitations prevent Bluetooth authentication mechanisms from being used, they should always be used. (**[LE-15] 4.15**).

Why might I not do this?

It is certainly possible for a product that supports Bluetooth pairing to not support an input mechanism that supports a yes/no selection, an input mechanism that support numeric entry, an output mechanism that supports the display, auditory playback or other means of communicating a six-digit number, or a combination of one or more of these capabilities. For devices that cannot support these mechanisms and cannot support an OOB mechanism (e.g. USB, NFC, IR), it becomes unavoidable to pair using an unauthenticated pairing mechanism. Care should be taken to ensure that an end-user understands the potential risks of pairing devices without support for authentication in range of a potential attacker (e.g. a public venue).

4.4 [LE-4] Use LE Secure Connections Pairing with LE Security Mode 1, Levels 3 and 2

Where Security Mode 1, Level 4 (LE Secure Connections authenticated MITM protection with 128-bit encryption key) is not supported due to hardware limitations, a device should still support LE Secure Connections pairing. If possible, it should support level 3 (authenticated pairing with encryption). If authenticated pairing is not supported due to hardware limitations (lack of input, output, or OOB exchange mechanism), level 2 can still be used, but offers no protection against a MITM attack during pairing.

Why should I do this?

If hardware limitations prohibit the use of security mode 1 level 4, using encrypted links is still valuable to ensure the confidentiality or integrity of the data transmitted. Levels 3 and 2 both still provide encrypted links, though level 3 is still preferable. If a device can support level 3, it should generally also be able to support level 4 and should.

What are the implications of doing this?

LE Secure connections pairing provides a more secure authentication mechanism than LE legacy pairing. The temporary key (TK) used in legacy pairing mode risks being brute-forced, which can result in loss of confidentiality of a recorded Bluetooth session. This lack of forward secrecy may permit an attacker to brute force the encryption long after the encrypted link closes.

When should I do this?

As long as the confidentiality and integrity of the data carried by a Bluetooth LE link needs to be protected, security mode 1 level 3 and 2 are the best remaining options once level 4 has been determined to be unsupported. Security mode 1 level 4 is a better alternative, but otherwise levels 3 and 2 should be used and if possible, LE Secure Connections should be enforced.

The remaining alternatives, security mode 1 level 1 and security mode 2, each have their own drawbacks. Security mode 1 level 1 and security mode 2 don't protect against eavesdropping. Security mode 2 is designed to protect message integrity but doesn't protect against certain types of replay attacks and does not protect against eavesdropping.

Why might I not do this?

Some services, for example, those providing read-only access to publicly accessible characteristics, may not require encrypted links. These services may use security mode 1 level 1. Care should be taken before using security mode 1 level 1, as it does not provide any guarantee for the data's recipient that a MITM hasn't modified the data. If the integrity of the data is at all important, use at least security mode 1 level 2 with LE Secure Connections (e.g. with JustWorks).

4.5 [LE-5] Support the OOB Mechanism for LE Legacy Pairing if LE Legacy Pairing Must Be Supported

The out-of-band (OOB) legacy pairing mechanism [3H, 2.3.5.4] provides the most resistant mechanism against MITM attacks of those pairing mechanisms supported in LE legacy pairing. When pairing devices both support an OOB transport that is itself resistant to MITM attacks (e.g. requires close proximity), then the temporary key (TK) exchanged in the OOB exchange has the maximal entropy available: 128 bits.

Why should I do this?

LE legacy pairing JustWorks [3H, 2.3.5.2] and Passkey [3H, 2.3.5.3] mechanisms both use less entropy in the temporary key (TK) than the OOB mechanism. The Passkey entry method has only 20 bits (effectively 19 are used[1]). The JustWorks method has 0 bits (none). OOB is more resistant against MITM attacks because the chances of the MITM attacker guessing the TK is significantly reduced, presuming the OOB mechanism is itself secure against attack and the TK used is not predictable.

An attacker who can brute force the TK and has a complete recording of the Bluetooth session has enough information to be able to determine the exchanged LTK and decrypt all the data transmitted between devices during the session [4].

[1] Passkey Entry [3, H, 2.3.5.3] supports passkeys from 000000 to 999999, which is represented by 2^19 < 999999 < 2^20 bits. 20 bits are effectively required to store all available Passkey values, but not all 2^20 options are permitted. 48576 combinations (2^20 - (999999 + 1)) are not actually valid, so not all 20 bits of entropy are fully used.

What are the implications of doing this?

Supporting an OOB mechanism in a design solely for the purposes of thwarting aa MITM attack during Bluetooth pairing rarely makes sense unless the OOB mechanism already exists as part of some other feature of the device. For a platform like a smartphone that often supports many viable out-of-band transports (WWAN, WiFi, NFC, Camera, Speaker/Microphone) a legacy peripheral supporting one or more of these mechanisms may already be supported. The choice of whether to support a specific OOB mechanism with Bluetooth technology depends on whether the technology is present in both devices and whether the protocol for exchanging the TK is known. In general, where support already exists, it makes sense to use it, otherwise, the introduction of support for a specific legacy OOB mechanism is likely not to afford much benefit.

When should I do this?

Where devices already deployed in the field that only support LE legacy pairing have an OOB mechanism that can be used by a newer design when pairing with the legacy device, that mechanism should be used. For newer designs, OOB can still be used, but the use of LE Secure Connections pairing should always be preferred over LE legacy pairing. Generally, this means that a device that commonly or exclusively pairs with a known peripheral that supports a known LE legacy pairing OOB mechanism should use that method preferentially. Otherwise, there are better or more broadly supported pairing mechanisms available.

Why might I not do this?

If your device does not support another mechanism that permits OOB communication, or the device with which you are pairing supports OOB, but not via a transport supported by your device, OOB will not be available. Even if not all devices with which your device will pair can support your device’s OOB mechanism(s), the OOB mechanism still presents the best protection against a MITM attacker and should still be used when available, if supported.

4.6 [LE-6] Use at least LE Security Mode 1 Level 2 (encryption with JustWorks pairing) for custom profiles instead of LE Security Mode 1 Level 1 (unencrypted)

In rare cases, an implementation may provide read-only access to data through a custom profile that doesn’t require data confidentiality. Even when it is not important for the transmitted data to be protected from eavesdropping, an encrypted link should still be used to ensure data integrity.

Why should I do this?

If there is any possibility that incorrect behaviors may result from receiving data that was manipulated by an attacker, encryption should be used to ensure that data is not modified by an injection attack. For example, a device offering access to read the current time in a public venue may still want to require an encrypted connection to ensure that a MITM cannot replace a PDU containing the time and cause incorrect behavior for the recipient device. This applies to any data that may be manipulated by a third-party.

If the data provided to a recipient is dependent upon the identity of the recipient, even if modification of the data would not result in harm to the recipient, there is also value in ensuring confidentiality of the data-stream to help ensure the privacy of the data’s recipient.

What are the implications of doing this?

Providing an encrypted link, even for publicly accessible data, helps ensure that the data is not modified by a malicious party. Using security mode 1 level 2 with LE secure connections and JustWorks pairing does not eliminate the potential interaction of an attacker, but the probability of success is greatly reduced. Bluetooth encryption does require an initial exchange of authentication information, which can take time (depending on the connection interval selected), and continuous execution of the block cipher algorithms where encrypting and decrypting payloads requires device resources, including computational time and energy. In most situations these effects are minor, but for LE devices that must provide access to their services extremely quickly or that have very limited energy stores (or extended battery-life requirements), the cost of authentication and encryption may be a significant consideration.

When should I do this?

Whenever the possibility exists that characteristics accessed from a remote device may result in some undesirable outcome if the data is manipulated or eavesdropped, the link used for this access should be encrypted. It is not necessary to require a fully authenticated link to gain at least some security advantage over transmitting data in the clear, so a device may pair and encrypt a link without requiring direct user interaction.

Why might I not do this?

If the data being transmitted is a general notification instead of a peer-to-peer exchange, advertising data can be used instead of requiring a connection. If the effect of receiving incorrect data in a payload as a result of deliberate interference is limited to a denial of service, it may be acceptable to use an unencrypted link. For example, a MITM hiding the presence of a feature in a characteristics database results in a temporary denial-of-service. A MITM indicating the presence of a feature in a characteristics database will result in errors when those characteristics are accessed by the attacked party, but no other side-effects should result. In both of these scenarios the effect of an injection attack by a MITM poses minimal risks, so the use of security mode 1 level 1 may be acceptable.

Devices with extremely limited ROM, computational resources, or available energy storage may find Diffie-Hellman Key Exchange or AES-CCM block ciphers to be power-hungry per unit data. In most cases, the LE controller will support the encryption algorithms and primitives and can perform the actual encryption on behalf of a code-space-limited host. Where the energy needs of encryption may become prohibitive, care must be taken to balance these competing requirements.

4.7 [LE-7] Periodically refresh the asymmetric key pair used in LE Secure Connections pairing.

The Bluetooth specification recommends that implementations refresh the public/private key pair used during the pairing procedure of LE Secure Connections after every pairing attempt [3.6]. Care must be taken when doing this to avoid leaking information about the private key, but this suggestion should be followed when possible.

Alternatively, the recommended schedule for key refresh can be used in lieu of a refresh every pairing attempt [2H, 5.1].

Why should I do this?

The public key exchanged in a LE secure connections pairing procedure does not normally provide information about the private key, but the pairing process takes input from the remote device and transforms it using the private key and provides a result. Since the data provided as input to this transformation can be generated by an attacker and the results observed, it becomes possible over repeated trials to gain information about the private key. Additionally, using the same key indefinitely can provide an attacker the time to attempt to brute force the private key. To ensure that both of these possible attack vectors are not feasible, it is valuable to occasionally refresh the key pair. For low power devices that do not have a means to readily support internal tracking of real-time, the pairing process itself is a useful trigger for a key refresh.

What are the implications of doing this?

The creation of a new public key can be a time and energy intensive process on a small embedded device. Where the battery life of a peripheral is key, an attacker can just as easily drain the battery by repeatedly failing encryption and forcing a key refresh (with different device addresses to avoid being locked out) as it can attempt to gain information about the key pair in use. Which is the more important consideration depends on the application.

When should I do this?

Where devices have sufficient computational resources or available energy, attempting the key refresh on each pairing is the most secure option, presuming that the time the chosen algorithm takes is constant or can be forced to appear to take a random time to an outside observer. In other cases, implementations should follow the recommendations for the minimum number of successes or failures to refresh after [3H, 2.3.66].

Why might I not do this?

Generating a key pair requires time and computational resources and is also potentially open to timing attacks, meaning that unless all key pairs generated take exactly the same time to produce, information might be gleaned about the key pair selected from the time the algorithm took. Generally, the key generation procedures in use on a host or in a controller should be hardened against such timing attacks (for example by not occurring synchronously during the pairing process), but a naive implementation risks leaking information. In cases where the energy usage of the key refresh cycle is a limiting factor for the utility of the device, it may be necessary to continue to use the same key for longer than is advised.

4.8 [LE-8] Avoid using LE security mode 2 levels 1 and 2.

Security mode 2 (levels 1 and 2) are intended to provide a viable unencrypted mechanism for exchanging data between two paired devices while ensuring data integrity. This is intended to make a device with very limited resources capable of performing attribute reads and accepting attribute writes without risk of an attacker modifying the data during transmission. Despite the advantages over security mode 1 level 1 unencrypted links, both available security mode 2 levels have flaws which make them poor alternatives to the encrypted connections already supported.

Why should I do this?

Neither security mode 2 level supports protection against eavesdropping. They are only a viable option when the confidentiality of attribute reads and writes is not important, but the integrity of the data is. This is achieved via data signing [3C, 10.4]. Even though this feature is provided for devices that might want to avoid introducing the overhead for full encryption, it still requires the exchange of a connection signature resolving key (CSRK) which requires pairing and an encrypted link (secure connections) for exchange.

Though the signature algorithm includes a running counter [3H, 2.4.5] to avoid trivial replay attacks, it is still possible to use a more sophisticated replay attack to perform an action that an attacker would be otherwise unable to reliably perform because of the presence of the unencrypted data informing the attacker of precisely what a packet is trying to do. This allows a sophisticated attacker to capture and selectively forward packets until it finds a packet changing the state of some characteristic it wishes to affect (e.g. the state of a toggled mechanism like a lock). The attacker can record this packet and suppress (interfere with) the original packet getting to its intended recipient, record and suppress a subsequent second attempt to send the same data (with a new MIC) but forward the original to the attacked device. If the counter hasn't increased between those messages, the attacked device will acknowledge the receipt of the message and the attacker now has a valid replay (the second attempt) in its possession which it can send to the attacked device and possibly leave it in an incorrect or insecure state (e.g. unlocked). Though non-trivial and not always viable, such attacks are not mitigated by security mode 2; therefore they represent a risk that does not exist when using security mode 1 (levels 2 through 4) for the same purpose (see Figure 6 -- Replay Attack (No Confirmation)). An implementation can use response packets, confirmations, or periodic no-operation control packets to defeat an attacker attempting such a replay attack.

What are the implications of doing this?

Avoiding security mode 2 level 1 and level 2 (or security mode 1 level 1) is the only way to ensure confidentiality of transmitted data (ATT reads or writes) without relying on a secondary encryption mechanism. Most data, if eavesdropped, provides an attacker either with restricted information about a user, undesirable details about the state of a system (are lights on, is the door locked, etc.), or information which can be used to compromise privacy (connected WiFi SSID, device name, etc.).

When should I do this?

Unless backward compatibility considerations demand support for security mode 2 levels 1 and 2, always use security mode 1 levels 2, 3, and 4 instead. Each provides additional security against eavesdroppers, levels 3 and 4 provide equivalent security against MITM attacks to security mode 2 level 2, and none provide an attacker a window for launching a successful replay attack.

Why might I not do this?

Only in cases where backward compatibility requirements or strict limitations in platform footprint limit the use of an encrypted stream should security mode 2 be used. Otherwise, it should be relegated to an interesting historical footnote. For applications that provide their own encryption mechanism, security mode 2 may be a lighter-weight mechanism for data transfer (avoiding the block cipher encryption stage), but incorporating a message authentication code in a custom protocol and relying on security mode 1 level 1 is expected to be a supported solution long-term.

4.9 [LE-9] Use private addressing modes and ensure that addresses used are not predictable.

In many application domains, the privacy of a user is an important consideration. In others, it may be an essential feature. Devices that advertise continually (or frequently) provide information to an attacker simply through the address used in the advertisement and any data included in the advertisement PDU. BLE provides non-resolvable private addresses (NRPA) that may be assigned to hide the identity of a device, and support for resolvable private addressing (RPA) which can automate the generation of random addresses while providing a way for a paired recipient also in possession of an identity resolving key (IRK) to identify that the address is produced by a known device.

Why should I do this?

A device that uses the same address each time it advertises (e.g. to establish a connection) provides a unique method of tracking that device. Once known, this identity can be used to identify an individual or to follow the activities of an individual using that device. For someone who either wishes or requires some public anonymity, using a device that provides this information represents a possible liability. Even devices without obvious privacy implications from their use may want to support device (or network) privacy [1A, 5.4.5] to provide for unforeseen use-cases.

What are the implications of doing this?

Using private addresses requires that the device generate non-predictable random addresses and periodically change addresses. For a device with an active connection, there isn't a standard way to change the private static address nor non-resolvable private address on the fly while maintaining the connection, unless a mechanism is used to update the recipient about the address change. Solving these issues for a BLE device supporting Core 4.0 and later is simply a matter of using resolvable private addressing. Support must exist on both the device using and the device resolving RPA addresses for the mechanism to operate seamlessly, but the mechanism is otherwise transparent. RPA requires the use of a shared resolving key (IRK) which must be exchanged during pairing/bonding. This does mean that only the remote device(s) paired with a device using RPAs will be able to identify it, but in most cases, this is the intended behavior.

The only major caveat with using resolvable and non-resolvable private addresses is that additional mechanisms are required in order to identify the device initially before initiating pairing. For peripherals that support a single paired device, this is usually a matter of providing metadata in an advertisement to identify the device only while it is unpaired. Obviously, this data must be of limited utility in identifying the device uniquely, otherwise, the privacy features of the device are only applicable when the device is paired. For platforms (e.g. smartphones), using RPAs is easier, as a scanning device doesn't usually need to identify itself to the advertising device (generally the peripheral).

When should I do this?

An implementation should support private addressing modes when that feature is supported by the host or controller, and support resolvable private addressing for paired LE devices in general. Unless there exists a specific impediment to using resolvable private addressing, it should be preferred over other addressing mechanisms in almost all cases. During initial advertising (i.e. before exchanging and using an IRK), a device should support NRPA or random static addresses (with some refresh mechanism) to avoid long-term tracking. This does require some support for known advertisement types in connectable or non-connectable advertisements to identify the device prior to completing pairing.

When possible, devices (especially peripherals) should not send advertisements while connections are established unless RPA is in use. In some cases, for example, where multiple paired devices are supported, this is not practical. In such cases, care should be taken to minimize the amount of time that a fixed device address is used.

Why might I not do this?

For devices without support for Core 4.1 or later, resolvable private addressing support isn't complete. Even when there is a need to support these devices, NRPAs or private static addresses may still be used to obfuscate the fixed identity of the device to potential attackers.

4.10 [LE-10] Obfuscate advertisement payloads when using private addressing.

Advertisements with data payloads are useful for finding and initially pairing/bonding and connecting with a device, but once paired, any unique information still included in an advertisement represents a method for tracking that device. When using private addresses (NRPA or RPA), including advertisement data that uniquely identifies a device (e.g. manufacturer-specific data, device-specific names) defeats the purpose of hiding the device through private addressing. Peripheral devices that are paired or bonded with a single platform should consider modifying any advertised data using a shared secret (i.e. using an shared encryption key with random data (salt) provided by intended recipient) or the current device address (where it is periodically updated) to obscure the source of the advertisements in a manner that helps to thwart tracking. This should be done only when interoperability is not an important consideration since any such data transformation will necessarily be application specific.

Why should I do this?

Devices that generate advertisements (connectable or non-connectable) with data may be tracked by an attacker observing the advertising data as easily as by observing the device address. Where the address is updated and private, but the advertisement data continues to be broadcast and contains identifiable information about the device, the attacker will simply prefer to use the advertised data instead of the device address as an alias to track the device.

In many cases, the information in an advertisement will not provide details about the device, but many cases exist where tracking information might be leaked these include scenarios like:

- The device name is unique and is included in advertisement data.
- Manufacturer-specific data is included and is either unique to the device or is unique enough that it can be used to reliably track the device.
- Values that decrease or increase reliably over time (e.g. battery life remaining) provide non-unique but still predictable details about a device (e.g. through modeling the approximate time the battery level is expected to decrement).
- The pattern of data in advertisements is dependent on the device and a knowledgeable attacker may decode the pattern to identify the device.

What are the implications of doing this?

There does not currently exist a standard method for obfuscating advertised data to maintain privacy. Where advertised data is unchanging but does not directly identify a specific device, tracking a device using this data requires that multiple devices of the same type not be in the same general area. It is advertised data (e.g. unique IDs) that identify a device directly that needs some form of obfuscation to avoid tracking by an attacker. It is necessary for both the advertising device and any applications consuming the advertisement to know how to reverse the chosen obfuscation process. A straightforward method for obfuscating the data is to encrypt it using a previously shared secret key and some periodically changing random input (salt) provided by the data's intended recipient(s), though it is recommended to never directly use an LTK for this purpose; doing so risks leaking information about the LTK. A custom key shared over an existing encrypted channel is a better candidate for encryption-based obfuscation, but care must be taken to not leak any information about the key through the encryption method used.

It is also important to avoid a transformation on the data that is based solely on the device's address since any attacker with knowledge of the algorithm used can easily reverse the process given the known device address and simply recover the original advertised payload.

When should I do this?

Generally, devices that are trying to remain anonymous, are paired with a device, and are designed to support only one pairing/bonding record should cease advertising data and instead exchange the same information via an encrypted connection. When a device needs to continue advertising data, device tracking will still remain an issue unless the information provided is sufficiently anonymous or is deliberately anonymized. A standard mechanism for providing this protection is not currently provided in the core specification, therefore devices wishing to avoid tracking and still advertise need to take steps to avoid providing information unique to the device or that creates a signature that mimics the observable or predictable behavior of the device. If this is not done, using private addressing will not be sufficient to hide the device's identity from a potential attacker (at least one interested in tracking the device instead of the identity of the device's user). Hence, anonymizing the data in advertisement payloads is required when using private addressing modes when both device privacy and advertising data payloads are also required.

An alternative is to produce advertisements with no data payloads (e.g. directed connectable advertisements). Hiding rather than obscuring data payloads avoids the need to manipulate the data. This is not always possible; for example, some identifying information needs to be provided when a device is first discovered by a user in order to establish the initial pairing/bonding relationship. It is possible to limit advertised data to scan-responses from a device using private addressing, but such scan-response data may also be observed by an attacker and should be similarly manipulated to avoid tracking.

Why might I not do this?

When interoperability requires that advertised data payloads be readable by any platform, not just a platform with awareness of the technique used to obfuscate the data, where privacy is not a significant consideration, or where device advertisements contain data which cannot be used to track the device, then obfuscating advertised data may not be required or supported. In most cases, a device should simply avoid data in advertisements when using privacy modes.

4.11 [LE-11] Ensure all random numbers used in cryptographic procedures are sourced from secure random number generators.

LE controllers are generally designed to provide their own random number generators and are expected to use cryptographically secure random number generation [2H, 2] conforming to FIPS SP800-22 [5]. Good implementations will also incorporate sources of real randomness (e.g. being re-seeded periodically using an internal measurement of thermal noise). Where a security manager or external libraries are used to generate random numbers that are used in one or more encryption or hashing functions (e.g. nonces) it is essential that these numbers not be predictable based upon the state of the device (e.g. time since initialization) and not be predictable based on the previous random numbers generated (e.g. as exchanged over the air).

Why should I do this?

Most cryptographic algorithms are only as secure as the unpredictability of the numbers used to generate keys and the random values used in various authentication procedures. Not all random number generators are intended to be secure; many are designed for efficiency or simplicity (e.g. a Mersenne Twister [6]) and are intrinsically insecure. Cryptographically Secure Pseudo-Random Number Generators (CSPRNG) implementations are available for many platforms, and the best use real entropy sources on the platform in order to continually reseed the random number sequence. For cryptographic algorithms, true CSPRNGs are needed.

What are the implications of doing this?

For implementations using non-secure random numbers, an attacker may use a known or measurable characteristic of the device to determine what random number is next in the sequence given information about the last few random numbers generated in a pairing procedure (for example). Being able to predict or force predicted random numbers to be used can immediately compromise the security of a device, as public/private key pairs may become predictable, authentication procedures may use predictable secrets, replay of encrypted payloads may become feasible and the security of the device may be readily compromised via access to a comparable device.

When should I do this?

Always use secure random number generators for all cryptographic algorithms. When it is unclear how a random number is sourced, do not use it for cryptographic procedures, rather incorporate a secure alternative. This shouldn't preclude the use of canned sources like the Bluetooth controller, but steps should be taken to ensure that a firmware implementation has not taken insecure shortcuts.

Why might I not do this?

Always do this. If your implementation uses existing libraries, request information about how the random number generation is performed and ensure shortcuts were not taken.

4.12 [LE-12] Do not bypass failing cryptographic procedures and don’t make a connection by other means.

Numerous normal and abnormal conditions may result in the failure of Bluetooth LE cryptographic procedures (e.g. operator error). Failure is not always an indication of the presence of a malicious actor, but failures at any stage may be an indication that an attacker is present and attempting to gain access through a vulnerability or through repeated trials. If cryptographic procedures fail (outside of controlled interoperability testing scenarios) falling back on less secure options for convenience is often the desired outcome for the attacker and invalidates the use of more secure alternatives.

Why should I do this?

An attacker is generally looking for an exploit, an implementation flaw, or an intrinsic flaw in an otherwise secure device. This is the means for gaining undesired access. When a cryptographic procedure (e.g. authentication) fails, it is essential that this failure is taken as a potential indication of an attack, even though it may be caused by an innocuous error. This doesn’t mean that a simple human error should trigger a device locking down (though for some applications this level of paranoia may be appropriate), but it does mean that an implementation should never accept a failure case and proceed with a subsequent step. This means that an implementation of pairing, bonding, encryption, MIC generation, authentication, or authorization mustn’t take shortcuts, and all of the possible vectors for failure should be both known and handled, usually by halting the procedure.

The temptation also exists to maximize user convenience by treating the repeated failures of a newer security mechanism as an indication that the newer features are not well supported and that an older mechanism may work better and should be tried. Though this may sometimes be true, in most cases having such security-downgrade mechanisms leaves an implementation open to attack on both newer and legacy security implementations. An attacker downgrading the security mechanism used (e.g. from LE Secure Connections to LE Legacy Pairing) is able to attack the device using the least secure feature supported.

What are the implications of doing this?

It is necessary to understand how errors may occur in the Bluetooth LE cryptographic procedures and terminate pairing procedures, authentication procedures, or encrypted connections whenever these errors occur. Some host implementations may manage these failure cases appropriately, but it is important to understand where there may be gaps. Failing to properly terminate connections or fail procedures when errors occur, may result in procedures continuing past the point where they are intended to work. In many cases, this will not result in security issues, but proceeding when there are errors can also result in security holes, for example:

- With numeric-compare authentication when an incorrect digit is entered or the user indicates the values do not match, the pairing procedure must halt or else a MITM (if present) may gain full access to the encrypted link.
- In LE secure connections, a failure due to an invalid public key (e.g. one that does not belong to the curve), if ignored, may permit an attacker to spoof a paired device if the procedure does not terminate.
- If a required minimum encryption key length is not met on a channel and the encrypted link is not terminated, an attacker may use that link to brute-force the data exchanged, given sufficient time and resources to launch a brute-force attack on the encryption key.

When should I do this?

Implementations should always terminate pairing/bonding, authentication or encryption procedures and terminate the associated links when errors occur. For example, a failure to establish a paired link with a device using OOB, numeric compare, or passkey authentication procedures should not result in JustWorks being attempted as an alternative.

Why might I not do this?

Unless interoperability requirements for an implementation demand that it integrate with multiple disparate designs with known authentication issues that require specific workarounds, implementations should not use an error in an authentication procedure as a trigger to try another (legacy) authentication model.

4.13 [LE-13] Configure LE Ping (when supported).

LE Ping is a feature that generally operates seamlessly at the controller layer. It ensures that an encrypted frame is sent on an active LE connection at regular intervals if no encrypted data is otherwise transmitted. In most cases, if LE Ping is supported between two controllers, it will simply be used, but can be useful to explicitly set the timeout used through the write-authenticated-payload-timeout command [4E, 7.3.94].

Why should I do this?

Using LE Ping ensures that an attacker cannot suppress a transmitted message while still maintaining the illusion that both devices are receiving data normally. In cases where a GATT command is used to perform a specific action, it is possible for an attacker to stop both the write-characteristic message and any response from being transmitted. This can leave the state of a write to a remote device unknown or indeterminate. If the connection is dropped, it may not be clear whether the command (or commit) was applied correctly. In some cases, this confirmation may be essential information. LE Ping will provide a notification if a valid encrypted payload is not sent after a period of time (the write authenticated payload timeout). If the host receives an event indicating the payload timeout has elapsed, it can then act to confirm that recent remote characteristics were properly updated, attempt to update them again or inform a user that an error has occurred (e.g. an unexpected disconnect). In most cases where it is essential that a specific update complete correctly, verification should be performed explicitly by reading back a status value to confirm a write occurred.

What are the implications of doing this?

LE Ping has little direct impact on LE encrypted connections aside from an additional packet being transmitted at periodic intervals if no other encrypted data is being transmitted. This does create some additional energy-use overhead that may not exist for a link using an otherwise large connection latency. The default write-authenticated-payload timeout is 30 seconds but can be shortened if confirmation that the link is still connected is required more often. It can also be lengthened if a packet every 30 seconds uses too much energy for the specific application.

When should I do this?

Unless energy use is being extremely tightly controlled, the default authenticated payload timeout provides for confirmation that an infrequently used active connection is still up and that the encrypted packet sequence remains uninterrupted within 30 seconds. In cases where confirmation of message receipt and link stability must be confirmed in less time, using the default authenticated payload timeout should be sufficient.

Why might I not do this?

The LE Ping procedure is generally supported or unsupported depending solely on whether the feature is supported in the controllers carrying the link. It is generally enabled automatically if it is supported. If the local controller supports LE Ping, it can be used even if the remote controller does not offer support (by using the encrypted unknown-PDU response from the remote controller). The authenticated payload timeout should be set to balance the power requirements for transmitting an occasional packet with the amount of time that should be permitted to pass before a device validates that the packets that have sent so far have actually been received by the remote device: shorter for time-sensitive or mission-critical applications, longer for energy-sensitive applications.

4.14 [LE-14] Do not store the Bluetooth security database in clear-text on devices that run multiple processes or provide remote memory access.

The Bluetooth Core specification does not specify how the EDIV to LTK association and the security key database are to be stored on a device (the EDIV applies to LE legacy pairing only). For embedded devices and devices without external network connectivity where physical access must be obtained to the device before such keys can be accessed, the options for storing this database are fairly unrestricted. In contrast, platforms where multiple processes run or multiple user identities are supported or where network access permits (theoretical) remote code execution risks, access restrictions to Bluetooth keys on the host need to be separately enforced.

Why should I do this?

A platform (e.g. a smartphone) where processes or the OS kernel are not properly sandboxed or protected against memory or storage access may introduce vectors for acquiring details about the LTK, IRK, or CSRK stored for a given remote device. Unless accessed deliberately for debugging purposes, an LTK stored in a log may also provide a method for compromising the security of the Bluetooth link by an attacker. If a networked application (e.g. a deliberate Trojan horse) can gain access to these keys, it can provide them to an attacker who may then actively decrypt and inject encrypted packets into the affected links and establish links to the remote devices as if were the remote paired device.

The solution to this problem is to design in protections against arbitrary access to the Bluetooth security database similar to those protections required for storing local passwords and user credentials. Don't store such data in the clear in long-term storage, don't allow access to the database outside of the kernel on platforms that permit isolation to privilege rings or that enforce context-based memory protections.

What are the implications of doing this?

Storing Bluetooth keys safely on platforms may add complexity or require that Bluetooth connections not be directly managed by applications. In most cases, any added complexity in a networked or multi-user design is justified.

When should I do this?

Whenever multiple processes could access the Bluetooth security database, especially where not all of these processes are strictly controlled, whenever memory protections do not strictly prohibit access to such a database (e.g. by getting a memory dump or accessing memory that hasn't been cleaned up after a context switch), or whenever a platform provides direct access to a network that may expose access to the outside world, the Bluetooth security database should be treated like any other restricted credential on the platform (though by no means necessarily stored in the same place).

Why might I not do this?

If a platform has no external network presence, does not permit memory access through its other interfaces (e.g. could a Bluetooth link be used to read memory or flash), does not support multiple user identities, has a single process (or monolithic kernel), or has no memory or context protections, then there isn't necessarily any advantage to protecting the Bluetooth security database against unintended access. Otherwise, care should always be taken that actors, even local actors, cannot acquire resources like the LTK unless this is explicitly required by a design (e.g. an explicit Bluetooth debugging mode).

4.15 [LE-15] Report the IO capabilities of the device [3H, 2.3.2] accurately to ensure the most secure authentication mechanism available is used.

The IO capabilities exchanged during pairing indicate what input and output capabilities (or OOB support) a device has. Indicating the most sophisticated input mechanism and output mechanism a device supports is necessary in order to use the most secure authentication mechanism available.

Why should I do this?

If a device supports a keyboard and a display, but the IO capabilities do not indicate this (e.g. to simplify pairing and force JustWorks), it becomes possible or easier for a MITM to launch a successful attack. Being able to use Numeric Compare, Passkey or OOB mechanisms for LE legacy pairing, and being able to use Passkey or Numeric Compare mechanisms for LE secure connections requires that the IO capabilities report that the pairing devices have input or output support to enable the use of one of these authentication mechanisms. Even a peripheral device with a single button can be used to confirm or deny an authentication request, so long as it also supports some method for displaying a number.

What are the implications of doing this?

Correctly indicating the supported input and output features of a device (or the user interface managing the Bluetooth pairing process) ensures that the most secure authentication mechanism can be selected when pairing or bonding.

When should I do this?

Always use the IO capability mapping that most closely matches the actual capabilities of your platform.

Why might I not do this?

If authentication could be used between your device and a remote device with sufficient I/O capability to support one of the LE legacy pairing or LE secure connections authentication methods, but is not used in order to simplify the user experience, then an attacker aware of this limitation can take advantage of the gap in order to either pairing directly with your device (e.g. if pairing is always enabled) or successfully interceded in pairing with a MITM attack. Do not bypass these mechanisms.

4.16 [LE-16] Validate that debug and sample keys are not used in production devices.

Some Bluetooth LE conformance tests (qualification) support the use of specific fixed keys for encryption-related services so that the correct behavior of certain encryption algorithms can be verified. It is essential that any fixed keys introduced for these conformance tests do not end up being used in an end-product.

Why should I do this?

A device with a fixed key that is known to an outside attacker is vulnerable to link decryption by an attacker. This makes Security Mode 1 Levels 2 through 4 equivalent to Level 1.

What are the implications of doing this?

Updating an implementation to use randomly generated public keys and LTKs requires a post-qualification modification. Care must be taken to ensure that this step is taken, though host stacks that support encryption should support these modes with minimal modification. For devices with fixed LTKs, it is still essential that each physical device has a different LTK.

When should I do this?

Always.

Why might I not do this?

Never ship a device with a publicly known or accessible LTK.

4.17 [LE-17] When using advertisements with obfuscated data with devices with private addresses (static or resolvable), always update advertising data if the device address is updated. Update both address and data values simultaneously.

Advertisements with data payloads may be used to track a device even when the device address changes periodically. Even if the data advertised does not directly identify a device, it may still be used to follow a change of the device's address if the data and the address don't change at the same time.

Why should I do this?

In situations where a device is the sole advertiser in an environment, following a change-of-address (which permits device tracking) may be trivial. In a public venue, where myriad addresses are present and devices constantly enter and leave sniffing range, a change of address can help obscure the identity of the device and halt tracking of the device. When the device is advertising a unique or recognizable data payload, a change of address without a concomitant change of advertising payload allows an attacker tracking that device to identify the change of address. Avoiding device tracking through this mechanism requires either changing the advertising data at the same time to a new value that does not relate to the original (in a way an attacker can discern) or not including data in the advertisements when the device is in a privacy mode.

What are the implications of doing this?

It is necessary to ensure that advertisement data changes whenever the device address changes, otherwise, the address change can be followed. To ensure this swap occurs so that it cannot be followed, it is necessary to update both the advertised data and the device address at the same time. This means disabling advertising temporarily, updating data payloads, updating the device address, and re-enabling advertisements (with the new device address). This technique is not directly supported by controllers; enabling resolvable private addressing with an automatic refresh [4E, 7.8.45] does not provide a mechanism that permits the advertisements to be updated without risking an advertisement being generated with the same data and the new address. In general, an implementation that wishes to ensure privacy and needs to advertise data needs to manage its own address updates.

There also does not currently exist a standard method for obscuring the data payloads in an advertisement such that an attacker cannot use them to track a device through its address changes. It is necessary for an implementation to either avoid traceable data or to use a custom mechanism to obfuscate the information (e.g. a transformation based on an exchanged IRK and a changing nonce in the advertised data) to ensure that only the intended recipients can decode it or identify its origin.

When should I do this?

Whenever the privacy of a device user is of significant concern and when the device also needs to generate advertisements with data payloads even while bonded or while in an actively connected state, it is necessary to both avoid the advertised data containing fixed trackable values (e.g. unique IDs) and to ensure that when the device address changes, so does the advertised data. Where the intent is solely to ensure that a publicly known value (e.g. unique ID) is not associated with the device, it is generally sufficient to ensure that the ID is not broadcast in an advertisement. If it is necessary to ensure that the identity of the device is not tracked by an attacker through address changes, then the advertisement data and the address must be updated simultaneously.

Why might I not do this?

When devices are not advertising data (e.g. using a directed connectable advertisement), the address update is generally sufficient to obscure the identity of the device. A resolvable private address will permit a remote paired/bonded device to establish a connection without revealing the identity of the device. It is necessary to synchronize the update to advertised data and the device address only in cases where it is important that the device's identity not be tracked through these transitions.

4.18 [LE-18] Require authorization for modifiable GATT characteristics or for restricted information in readable GATT characteristics.

(**[LE-2] 4.2**) recommends that authentication and encryption be enabled for GATT characteristics that can be remotely modified. LE also permits the introduction of an external authorization mechanism that may be enforced before access to a characteristic is permitted. Authorization requires that a non Bluetooth mechanism (e.g. a login) be used to authorize access to an attribute [3C, 10.5], [3F, 4] and [3G, 8.2].

Why should I do this?

Bluetooth authentication and encryption ensure that a third party without physical access to the device cannot eavesdrop on a devices' communications and cannot write to attributes without pairing, but an attacker with physical access to a device can usually pair with the device and gain access to the information on the device or write attributes directly. To ensure that physical access does not imply access to the data of the device, it is necessary to use an application-level authorization mechanism. GATT characteristics include a flag to indicate that authorization is required in order to access an attribute [3F, 3.2.5].

What are the implications of doing this?

Authorization requires the introduction of an application-level mechanism to configure and manage whether a remote device is authorized. An insecure approach to enabling authorization is to require a fixed password to be known and entered before access is granted. A more secure approach is to add a mechanism to join a device to a security domain (e.g. a system analogous to a RADIUS [7] server) and provide credentials to the device that must be proven (e.g. matched) whenever an application reconnects. A configurable password is a common mechanism for enforcing this additional authorization, but other mechanisms include a shared secret only accessible to a platform accessing a peripheral are also viable. These authorization methods currently require non-standard mechanisms outside the scope of Bluetooth specifications. This is slated to improve with the introduction of the Authorization Control Profile and Service (ACP/ACS).

When should I do this?

Whenever a device's behaviors or information should be restricted even if physical access to the device is obtained, the authorization should be required for its attributes, especially writable attributes which may affect the behavior or condition of the device. For example, a medical device that monitors and records blood glucose levels should not offer up its records to any device that pairs with it. Such access should require some additional level of authorization for the protection of an individual using the device.

Why might I not do this?

Using authorization requires support for a common authorization mechanism on both devices involved in confirming authorization and may interfere with the interoperability of solutions using a peripheral until authorization standards are agreed upon. Login credentials, password management, storage and validation all have their own vectors for compromise which must be managed correctly to ensure that the device is sufficiently secure for its intended use.

4.19 [LE-19] Require authentication, encryption and authorization for all GATT characteristics.

(**[LE-2] 4.2**) recommends that authentication and encryption be enabled for GATT characteristics that can be remotely modified. It is also frequently valuable to introduce authentication, encryption, and (where supported) authorization for read-only attributes or for all characteristics associated with a specific profile.

Why should I do this?

Enabling common security features for all GATT characteristics in a profile ensures that those properties cannot be accessed without first gaining an encrypted, authenticated or authorized link. Although some attributes may not require this level of protection, it is unlikely that a profile that requires encryption, authentication and authorization for its writable attributes will require no security whatsoever for read access to other characteristic values in the same profile. The attribute permissions are constructed to permit maximum flexibility, but in most cases the attributes constituting a profile should not allow different levels of access, because the underlying service should only be accessible to a device or user when the appropriate level of protection has been reached. To require that a property be encrypted to be written, but not to be read opens up a window for an attacker to inject a packet and misrepresent the state of a device's characteristic on a subsequent connection. If a door was locked (authorized attribute) and confirmed to be locked, then the connection is dropped and some time later the locked status is read back without even requiring encryption, is the door still really locked (i.e. if a MITM attacker is providing a spoofed 'locked' status)? These considerations become moot when the footprint for security for devices is consistent.

What are the implications of doing this?

Assigning the same permissions to all attributes in a service ensures that access to a service does not occur at different levels of access at different times. In some cases, this may add complexity to a user interface or be undesirable when it is truly intended for some attributes to be accessed with fewer permissions than others. In most cases, this is simply good practice and avoids subtle side effects that an attacker might exploit (reads and writes being subject to different types of attack for different connections at different times on the same service).

When should I do this?

Whenever a service has attributes which require permissions, either all the permissions should be set the same, or care should be taken to ensure that permissions are exactly what is required for the purpose of each attribute. Generally, if authorization is required to access writable attributes, it's important to also require authentication and encryption, and for any related readable attributes, it's a good idea to include authentication and encryption attributes even if authorization is not required.

Why might I not do this?

Certain services, especially custom services, may have very specific requirements for certain attributes (e.g. public access) and a blanket set of permissions may not be appropriate for the intended operation of all attributes in a service. In most cases, a service provides access to a specific feature of a device and its security attributes should be consistent or at least logically arranged.

4.20 [LE-20] Change the EDIV for legacy pairing by refreshing pairing when device privacy is required.

LE Legacy Pairing includes an encryption diversifier field (EDIV) in the data exchanged between pairing devices. This random identifier is used in part to provide an index to a stored LTK in a device's key database. In LE Secure Connections, the EDIV is no longer used and is always assigned 0 [3H, 2.4.4].

Because the EDIV is unique for a given device pairing, it represents a method for tracking the identity of a device while the underlying LTK remains unchanged. A long-term bonding between a device supporting Core 4.2 or later and a device supporting 4.1 or earlier will result in an EDIV being generated and provided with each encryption establishment [6B, 2.4.2.4].

If device privacy is an essential feature of a device and private addressing modes are used, the presence of the EDIV can be used to track the paired device regardless. The EDIV should be periodically updated to prevent long-term tracking. Doing this requires establishing a new pairing/bonding between the devices.

Why should I do this?

Where device privacy is essential and a device supporting LE legacy pairing is in use, the encryption diversifier may be used by an attacker to identify a device when a connection is established. When it is essential that privacy is maintained, it is valuable to not maintain the same EDIV/LTK pair indefinitely. Re-pairing devices is sufficient to trigger the generation and storage of a new EDIV. If a device does not bond, the EDIV will not be stored.

What are the implications of doing this?

Deliberately refreshing device pairing requires that devices occasionally delete their stored EDIV and LTK and initiate a new pairing procedure. If a device never bonds, this should occur each time encryption is required. Whether pairing is initiated by forcing the deletion of credentials or whether it occurs every time encryption comes up, the user experience may be less than optimal if a pairing procedure is required at random intervals.

When should I do this?

Refreshing pairing to ensure privacy should only be used in very specific circumstances where both LE legacy pairing and device privacy is required. In most cases, this issue can be circumvented by using LE Secure Connections.

Why might I not do this?

With LE legacy pairing, the process of pairing/bonding, if the entropy of the pairing method is low, may introduce the ability of an attacker to compromise the confidentiality or integrity of an encrypted link, which is usually a more severe attack than an attack on the privacy of a link. Only in cases where the privacy of a pair of devices relying on LE legacy pairing is of paramount importance should a periodic refresh of device pairing/bonding even be considered.

4.21 [LE-21] Do not share LTKs among multiple remote devices.

The long-term key, since it used to derive the encryption keys used when encryption channels come up, can theoretically be used to construct a naive multicast or broadcast encryption mechanism among multiple devices, or provide a shared pairing between more than two devices. Doing this requires sharing the key via a mechanism that may not be as secure as the original key exchange, and immediately opens up implementation-specific attacks on the new key exchange procedure (including an attacker triggering it). Doing this also makes every device with that LTK a viable target for a trash-can or MITM attack, since access to one of the devices with the LTK allows one to spoof and/or share that same LTK with even more devices or attackers.

Why should I do this?

Sharing a single symmetric key amongst multiple devices risks each one of those devices becoming a vector for breaking the encryption of any of the other devices, especially if physical access to the device allows an attacker to acquire the key directly. The security domain of a Bluetooth link is designed to encompass the connection between two peer devices. To extend it to multiple peer connections fails to introduce common protections required for distributed keys to the other devices, including key revocation (block) and device-level security (any device with the key can decrypt any link between any two devices with that same key).

What are the implications of doing this?

Ensuring that each peer-to-peer link has its own unique LTK can increase the number of total pairings required and does not scale to a fully-connected network of devices. Nevertheless, it is important to avoid using keys to create a network security architecture when there is no protection against common attacks on nodes in a shared security domain. Secure group multicast encryption mechanisms exist in Bluetooth Mesh and LE Broadcast Isochronous Groups and Streams, and should be preferred over a custom solution based on existing Bluetooth security mechanisms.

When should I do this?

Always store the LTK on only the two peer devices involved in a bonding. Storing the key on more than two devices compromises the security relationship between the two devices. Even transferring the key provides a method for attacking the security of a Bluetooth bond. Technologies exist (including Bluetooth mesh technology) which provide solutions for the problems inherent in secure multiple links between devices in a small network.

Why might I not do this?

Never share the LTK amongst multiple devices. If group broadcast or multicast communications are required using Bluetooth LE, consider using the Mesh profile specification to manage the group relationship.

4.22 [LE-22] LE Isochronous Broadcast should use Mode 3 Level 3.

LE Broadcast Isochronous Groups (BIG) use LE Security Mode 3 to secure the shared key (broadcast code [3C, 3.2.6.3]) used for encrypting LE isochronous broadcast PDUs [3C, 10.2.5]. Security Mode 3 Level 3 requires authentication when sharing the broadcast code.

Why should I do this?

Using an authenticated method to exchange the broadcast code when introducing a device into a BIG is intended to ensure that only the device that was supposed to receive the broadcast code does so. If a MITM is present when the broadcast code is shared and an authenticated method is not used, the broadcast code may be exchanged with or generated by the MITM which will then have access to decrypt all encrypted data exchanged within that BIG.

What are the implications of doing this?

As of Core 5.3, the method for performing authentication in security mode 3 level 3 is not specified within the core specification. Following this recommendation may confer additional security in the passing of a broadcast code, but only if the implementation of the authentication mechanism is sound. In general, for a minimally secure authentication technique, ensuring that the sharing of a broadcast code only occurs with an authenticated party will ensure that an attacker (e.g. MITM) will not gain access to the broadcast code.

When should I do this?

When the data transmitted within a Broadcast Isochronous Group (BIG) carries information which should be kept private (e.g. contains audio representing a personal conversation), using authentication to protect sharing of the broadcast code can help ensure that the broadcast code is not eavesdropped by an attacker and used to decrypt the session. As possession of the broadcast code is sufficient to decode BIG, this mechanism should be considered necessary to avoid eavesdropping.

Why might I not do this?

In cases where the Broadcast Isochronous Group (BIG) carries information intended for public consumption, even where access is intended to be limited by a broadcast code (e.g. public announcements), there may not be an advantage to authenticating the broadcast code even if there is value in using one. In such cases, security mode 3 level 2 may be appropriate.

4.23 [LE-23] Use a strong Broadcast-Code for LE Isochronous Broadcast

Why should I do this?

The broadcast code is a 128-bit key that is used to encrypt Broadcast Isochronous Group (BIG) and Broadcast Isochronous Stream (BIS) communications [3C, 10.9]. A broadcast code is enforced by specification to have a UTF-8 string representation at the device UI layer [3C, 3.2.6.3] of no less than 4 octets. In theory this means that the shortest permitted broadcast code may still have billions of possible values. In practice, a user is likely to use a smaller subset of those values (e.g. '0000'). Since knowledge of the broadcast code and the advertised GIV and GSKD [6B, 4.4.6.10] is sufficient to decode a broadcast channel, the broadcast channel can be brute forced by an eavesdropper by trying plausible broadcast codes (from short codes to longer codes). Therefore, it is important to use a long random string for the broadcast code, ideally a string that cannot be guessed by password cracking techniques. In contrast to a password system, which usually has a limit on attempts or a back-off time to prevent an attacker from rapidly trying unlimited numbers of passwords, an eavesdropper in possession of a recorded isochronous broadcast session may attempt a brute force attack on the encrypted BIG or BIS at any speed it is capable of.

What are the implications of doing this?

Using a broadcast code with a sufficiently long string representation will make the code's value more difficult to brute force. It will also make it more difficult to remember and enter when it is being used to manually introduce devices into the same group. Where an implementation supports sharing of the broadcast code over another encrypted channel, this becomes less burdensome on the user.

A short broadcast code may be appropriate for certain use cases, but it does not offer much additional protection against eavesdropping beyond what an unencrypted session provides. Encryption can provide protection against packet modification while an attacker searches for the broadcast code, so an encrypted session still has value for message integrity, even if an attacker could plausibly decode a recorded session later.

When should I do this?

So long as the data carried by BIG or BIS should not be eavesdropped or manipulated by a third party, a strong (long and random) broadcast code should always be used. Changing the code occasionally is also valuable, presuming that there exists a ready mechanism for exchanging the code again (not using the original code to encrypt the new code).

Why might I not do this?

If the broadcast channel is not encrypted or is encrypted but is carrying publicly accessible media (for example airport announcements for hearing aid users), it may be perfectly reasonable to use an easy-to-remember broadcast code. Such a code would not need to protect against eavesdropping since the information is readily available. A code should still be used (and changed periodically) to keep an attacker from trying to inject fake data (e.g. making fake announcements).

4.24 [LE-24] Limit the use of scannable advertisements and active scanning when operating in privacy modes.

Why should I do this?

Devices operating in device or network privacy modes, particularly when using Resolvable Private Addresses (RPAs), can be tracked both by how they respond to devices in their Filter Accept List and how they do not respond to devices not in their Filter Accept List. If a device with an address not previously discovered by or resolvable to an eavesdropper in range is observed to a respond scannable advertisement with a scan request and the advertiser then responds with a scan response but not to other scannable advertisements, it is immediately apparent that those two devices may have a relationship (i.e., that they are bonded or have at least shared an Identity Resolving Key (IRK) after pairing). If the RPA of one of the devices changes, but the other device's RPA does not change simultaneously, the continuing pattern of message exchanges between the devices can be used to track the two devices as a pair. They become recognizable as a pair.

This is possible because an accept-list used to filter which devices to respond to provides information about that device depending on when a scan request occurs and when it does not. A device sending a scannable advertisement can expect a scan request from a device which knows about it. Its own response to that request based on its own Filter Accept List can disclose its relationship with the scanner. An attacker could passively identify a device or it could actively trigger a device to identify itself using a previously observed RPA without knowing anything else about the device. To avoid having a third party identify the pair of devices using their communications patterns or use known RPAs to trigger specific devices to respond, it is necessary to suppress these patterns.

Disabling active scanning and scannable advertisements for the Central and Peripheral role devices (respectively) can help avoid response patterns emerging and can avoid pairs of devices being identified by a third party.

What are the implications of doing this?

Disabling active scanning will stop a device operating in the scanning state from making scan requests, but it will also disable scan responses that a remote device would generate in response to those scan requests. Disabling scannable advertisements will limit the legacy advertisements that a device can transmit, including connectable advertisements for devices not using extended scannable advertisements.

The tradeoff with using limited advertisement types and scanning modes affects when device discovery and device connectivity are available. In most cases, Peripheral devices passively advertising but not yet paired or bonded with a Central have not yet enabled a privacy mode so they are trackable, but this can be best avoided by pairing/bonding and negotiating an IRK. Once the IRK is exchanged and privacy is in use, both Central and Peripheral role devices should take care to minimize patterns which could enable tracking.

When should I do this?

Central-role devices that are in a device-discovery mode or in the process of scanning for connectable or scannable advertisements could expose both devices to tracking by a third party. Devices that are able to restrict advertisements to extended advertisements in both Peripheral and Central roles have more options for using non-scannable advertisements. For legacy advertising and scanning, the use of non-scannable advertisements and passive scanning is appropriate to avoid tracking when not attempting to establish connections (for legacy advertisements).

Peripherals may need to provide scannable advertisements while in discoverable mode, but once paired/bonded with a remote central and using device or network privacy, connections or directed advertisements can be used to transmit data payloads intended for the Central-role device. If a Peripheral is designed to permit multiple connections this may not be practical, but it should still be possible to limit the time-duration that the Peripheral is discoverable.

Devices using non-scannable directed advertisements should consider using low-duty-cycle directed advertising to avoid impacting power consumption.

Why might I not do this?

If scan responses contain data payloads that are needed to support a particular application, limiting the duration of active scanning may be undesirable. If a pair of devices operating in the Central and Peripheral roles support only legacy connectable advertisements, scan requests may still be generated and may still pose a privacy risk. Generally, devices that need to support privacy should disable support for scannable advertisements and active scanning when those features are not required. Devices with no support for privacy modes or private advertisements, or those with advertising data that is intrinsically trackable (e.g., unique values) may not need these restrictions since they will generally be trackable with or without such restrictions.

4.25 [LE-25] Support Bond Management Service (BMS) in devices that support bonding to help avoid device tracking after repurposing or disposal.

Why should I do this?

Devices that establish a permanent pairing relationship (Bonding) with a remote device must necessarily store an identifiable address of the remote bonded device in the Bluetooth Security Database, usually the public or static random address of the bonded peer. Normally, when a remote peer erases its bonding, the local device is not aware of this change, and its local bonding data is left intact.

For example, if a Central-role device has a public identity address (e.g., a BR/EDR BD_ADDR) that it has shared with a Peripheral, erasing or 'forgetting' the bonding from just the Central-role device normally leaves intact the address and potentially other identifying information about the Central-role device on the Peripheral. If the Peripheral device is then sold, repurposed, recycled or disposed of in a non-destructive manner, it may be possible to recover the identity of the peer from the security database. The ease of doing this will depend on the architecture of the storage medium where the Bluetooth security database is stored.

If both the Central and Peripheral role devices support the Bond Management Service (BMS), and at minimum those BMS features that enable support for erasure of the bond management record, then the BMS can be employed to delete the Peripheral's bond by the Central-role device explicitly when those devices are connected and collocated when the bonding is erased. This permits both devices to reliably delete their bonding records at the same time, avoiding the retention of trackable information. The ability to recover previously stored data in the security database will depend on storage architecture on a given device.

What are the implications of doing this?

Support for and use of the Bond Manager Service allows a device's bonds to be managed by its bonded peer, normally the peer that established the original bonding (generally the Central-role device). Providing a peer with the ability to delete a bonding carries a small risk of denial-of-service by a device or individual spoofing the peer but permits a bonding to be deleted in a manner which preserves privacy for devices that are repurposed, resold, or otherwise accessed after an intended disposal.

When should I do this?

When device privacy is essential, it is important to not only eliminate trackable information while a device is in use, but also to protect against deliberate tracking through the information stored by its peers. A decommissioned device usually presents a low risk of tracking, but the risk that exists is also easily mitigated through deletion of a bond using the Bond Management Service. Devices that wish to maximize the privacy of their users should consider implementing the BMS. A device should also support the BMS when its peers are anticipated to support deletion of bonds through the BMS.

Why might I not do this?

If a device has a method for permanently erasing the Bluetooth security database (e.g., a factory reset), this feature can be used instead of the Bond Management Service to permanently erase any record of the relationship between devices. It may still be useful to support the BMS to avoid an end-user needing to understand the need to take this step to help protect their privacy.

If a device is designed such that access to the Bluetooth security database is effectively impossible (e.g., where it would require de-lidding and scanning electron microscopy to read or the information is stored/used permanently on a SoC employing dedicated hardware encryption), access to this information becomes sufficiently impractical that explicit deletion isn't necessary.

If the Central-role devices anticipated to be used with a Peripheral do not support the BMS, then the BMS service on the device will not be exercised, and its presence is irrelevant.

The BMS supports but does not require the use of authorization to validate that a peer is authorized to delete the bond. Absence of authorization restrictions generally permits any paired/bonded device to delete a bonding, regardless of whether it originated the bonding being deleted. This risks a Denial-of-Service on the bonded devices by another paired or bonded peer. Authorization, though supported, is a vendor-specific behavior of the BMS.

4.26 [LE-26] For devices that use RSSI to estimate range to a peer device, avoid using the RSSI as the sole indicator of physical distance.

Why should I do this?

If a device requires a measurement of physical distance to its peer, it may use the received signal strength indication (RSSI) from the remote peer to approximate the range between the remote transmitter and the local receiver. Though this value is often difficult to calibrate, it is usually possible to bound the distance from a device to its peer when the maximum power output of the peer is known. When power measurement is above a certain threshold, the peer device must be at a certain minimum distance, though it may be closer.

Even when the accuracy of this measurement is high, an attacker may still use a relay attack (see 'Relay Attack' in Common Attacks and Mitigations above) to raise the measured RSSI, making the device appear closer. Without a secondary source of position information or other mitigations against a relay attack, a high RSSI value may provide no information about the proximity of the peer transmitter to the local radio. For devices that affect a specific behavior when proximal to one other, this can be used to fake the proximity value and fool the devices into completing an action when the devices are not physically close (or alternately not sufficiently far from one another). So long as a confirmation action is required, and this confirmation action isn't itself vulnerable to a MITM attack, an attack on the accuracy of the range value may not be relevant, but it could still be used to launch a brute-force attack against a locking mechanism (for example a vehicle key-fob or door lock) while its owner is unaware.

The use of RSSI as a signal-strength indication to a user is less likely to be manipulated by an attacker in a manner that can cause harm. But similarly, using RSSI as a reliable indicator of presence is also not reliable, as the address of the remote peer may simply be spoofed to make it appear to be in range.

What are the implications of doing this?

Using only the RSSI to measure range requires no additional hardware beyond the Bluetooth LE radio but introducing a secondary source of position information or introducing other range measurement techniques is still necessary to avoid a relay attacker spoofing the RSSI value undetectably. Introducing additional range-measurement features requires additional hardware radios (e.g. GPS) or further methods for determining range using signal characteristics and may impose additional cost or complexity.

When should I do this?

Whenever the range between peers is used to trigger some action, especially where such action exposes a user resource (for example a vehicle or door) to unauthorized access or use, some secondary confirmation of the accuracy of the range should be used to avoid spoofing the range to achieve unauthorized access.

Why might I not do this?

For devices that require some measurement of range but cannot support any methods beyond the measurement of RSSI, there may be no good option for mitigating relay attacks. Otherwise, RSSI should not be considered a reliable method of measuring range (on its own) and should not be used in exclusion for this purpose. If an explicit action is still required by the user (e.g. the pressing of an unlock button) to perform an action, and that action cannot be performed by an attacker using a replay attack (see 'Replay Attacks' in Common Attacks and Mitigations above), then mitigating the risks is still advisable, but may not be strictly necessary.

5 Bluetooth® Classic (BR/EDR) Recommendations and Best Practices

5.1 [BR-EDR-1] Use Secure-Connections-Only Mode.

Bluetooth BR/EDR supports a number of different pairing, authentication and encryption modes in order to support various legacy devices. All legacy security modes have known security limitations. To ensure that modern Bluetooth devices using BR/EDR use FIPS-approved algorithms (including AES-CCM encryption and Elliptic Curve Diffie-Hellman key exchange), 16-octet encryption keys, and authenticated secure simple pairing, a device should enable secure connections only mode [1A, 5.3] and enforce this.

Why should I do this?

BR/EDR legacy pairing, legacy encryption, and key-length negotiations all have known attack vectors which may result in compromise to the confidentiality or integrity of an encrypted link. Resolving these limitations requires the use of BR/EDR Security Mode 4 Level 4. To ensure that an attacker cannot downgrade the security of a link by convincing a device to use a less secure legacy standard (e.g. by spoofing the unencrypted feature mask), hosts using secure-connection-only mode will refuse connections to those devices supporting only those legacy pairing and encryption procedures [3C, 5.2.2].

What are the implications of doing this?

Using secure connections only mode ensures that FIPS-compliant algorithms are used by the controller during pairing and encryption and ensures that the encryption key length is not reduced. This is desirable when security mode 4 level 4 is supported on both devices, but this can cause issues when legacy BR/EDR devices that do not support these modes are used because secure-connection-only mode will act to prohibit these less secure connections from being established [2.2]. Thus, increased security comes at the cost of reduced compatibility, and the balance between the two options must be considered carefully.

When should I do this?

When it is important that the integrity or confidentiality of an encrypted link not be compromised, a device, especially a peripheral that carries important confidential information (e.g. a keyboard into which user passwords are typed), should always enforce secure-connections-only mode. This ensures that the security of the link cannot be downgraded by an attacker but does limit the supported platforms to those that support security mode 4 level 4 (4.1 and later).

Why might I not do this?

If backward compatibility is an essential feature of a device or platform, security mode 4 level 4 links can still be established but enforcing security-connections-only mode and disallowing other modes may not be a viable option. Under normal circumstances, two devices that support security mode 4 level 4 but are not enforcing secure-connections-only mode will still use the most secure pairing and encryption algorithms, but there still exists a plausible attack vector where an attacker may downgrade the security of the link and establish an encrypted link that is valid but permits the attacker to succeed in another attack.

5.2 [BR-EDR-2] Use Secure Simple Pairing if Secure-Connections-Only Mode can't be enabled.

BR/EDR legacy pairing uses a SAFER+/128-derived algorithm [2H, 6.3] which is not considered FIPS compliant [8]. There are some plausible key-schedule weaknesses in SAFER+/256 [9], but these do not lead directly to plausible attacks on the Bluetooth pairing procedure. The issues with legacy pairing are related more to the use of fixed pin codes, which generally results in links without forward secrecy (since knowledge of the pin and a capture of the pairing procedure can result in breaking the encryption of the link after the fact). Secure simple pairing introduces Elliptic Curve Diffie-Hellman key exchange and mechanisms for authenticating a link beyond the use of a pin code. Using secure simple pairing provides reliable authentication methods like Passkey and Numeric Compare [3C, 5.2.2.8] that provide consistent MITM protection, ensuring that a MITM attacker will fail to intercede in the key exchange procedure with a probability of 99.9999% or greater.

Why should I do this?

The secure simple pairing feature introduces authenticated and non-authenticated (JustWorks) pairing procedures that provide forward secrecy and avoid fixed pin codes. Secure simple pairing protects against an attacker decoding the link after the pairing procedure completes by brute-forcing the pin code used for pairing. This provides much better protection against attacks on the confidentiality of a link and should always be preferred.

What are the implications of doing this?

Secure simple pairing support is required for BR/EDR devices supporting Core 2.1+EDR and later. Enabling secure simple pairing in the controller forces secure simple pairing to be used with devices that support it, so there is little additional effort in providing support.

When should I do this?

Devices supporting just legacy pairing (before 2.0+EDR) are using a withdrawn Bluetooth specification. Devices after 2.1+EDR support secure simple pairing which should be preferred whenever it is available on a remote device. It is still possible to need to establish a connection to a device that does not support secure simple pairing, but in most applications, this should be considered a rare requirement. If there are risks associated with loss of data confidentiality, an implementation should consider enforcing secure simple pairing by disallowing legacy pairing or by implementing secure connections only mode.

Why might I not do this?

Only if a remote device is pre 2.1+EDR should there be any reason to rely on legacy pairing, but if support for legacy devices is required (e.g. in motor-vehicle applications where the Bluetooth module may be as old as the vehicle) there may be no practical choice other than to provide interoperability. In general, legacy pairing must be supported by a BR/EDR or dual-mode Bluetooth controller, but that does not imply than an application must accept pairing requests using pin codes. Care should be taken in deciding to permit legacy pairing.

5.3 [BR-EDR-3] Enforce the maximum encryption key size supported.

The BR/EDR link manager protocol (LMP) uses an exchange between devices initiating encryption to determine the length of the encryption key to use when encrypting the link. This negotiation was provided in part to support devices that must operate in jurisdictions where the highest supported Bluetooth encryption strength is not permitted. This exchange is still supported for backward compatibility reasons. In general, most controllers will request the maximum supported encryption key size, but this value will not always be supported by both devices. An application requiring a specific encryption strength should verify that the selected encryption key strength has been negotiated, both to ensure that an attacker cannot decrypt the link by brute force methods, and to ensure that an attacker has not modified the key size exchange in order to reduce the negotiated key length.

Why should I do this?

As of Bluetooth Core 5.1, it is permissible for a device to negotiate and use an encryption key size of as little as 1 octet. This is not recommended and may be restricted by post 5.1 controllers, but it is possible to have a 1 octet key length negotiated deliberately by a paired device or by an attacker interfering with the key-size exchange. Since only 256 trials are required in order to brute-force an encrypted link using an encryption key length of only 1 octet, this offers no additional protection over an unencrypted link.

To ensure that an encrypted link cannot be trivially decrypted, applications requiring strong encryption should validate that the link between devices carrying the data of its profile is using an encryption key length of sufficient strength. In general, this should be the maximum size supported (16 octets as of Core 5.3).

What are the implications of doing this?

Ensuring that the encryption key length is sufficient for a given application ensures that the BR/EDR link is not operating with a reduced encryption key strength. The strength required for a specific use case depends on the data being carried. For an A2DP profile, encryption strength is not necessarily an important consideration (though it may be if carrying voice), but a HID keyboard profile may be carrying user passwords and should not permit encryption key strength reductions.

When should I do this?

Whenever the encryption strength of a BR/EDR link is an essential feature of the connection, and a profile in use relies on the Bluetooth baseband link to provide security and does not support further encryption (e.g. SSL over BR/EDR PAN), care should be taken at the host or application level to ensure that the encryption key length is sufficient for the profile's purpose. If not, the link should not be used to carry sensitive data. How to obtain a link with a sufficient encryption key length or tear down and re-establish encryption if a link is not using a strong enough encryption key are both considered application-specific behaviors.

Why might I not do this?

Where a remote device does not support or chooses not to support the maximum permitted encryption key length, it is possible for an encrypted link with that remote device to never be established with a sufficient encryption key length. This may disallow the use of a device that is needed for a specific application if a profile-level connection is not permitted when the link encryption size can never be sufficient. In situations where the set of devices supported by a platform is known and the encryption strength can be controlled, or when the encryption key size must be greater than a set value to protect confidential information, it makes sense to restrict the encryption key length. Otherwise, it may not be possible to enforce this restriction and still have the implementation be interoperable with all devices a user may wish to use.

5.4 [BR-EDR-4] Disable discoverable-mode when not actively pairing.

A device is in discoverable mode [3C, 4.1] when it responds to an inquiry with an inquiry response or extended inquiry response [3C, 8]. A device may enter limited-discovery mode when it wants to enable the device to be discovered for a limited period of time, or it may enter general-discovery mode when it wishes to be discoverable indefinitely (at least until made non-discoverable). While discoverable, a device responds to INQUIRY_SCAN packets it receives with an FHS packet containing the device address. This leaks the Bluetooth device address of the discoverable device to anyone listening, and of course the device may be explicitly discovered by any other device within range. Because the Bluetooth device address (BD_ADDR) of a BR/EDR controller is a fixed value, this represents a significant vector for device tracking.

[NOTE: Some devices may support Bluetooth address assignment via a vendor-specific command. This may be used to provide some additional privacy at the cost of compatibility. BR/EDR hosts are not generally expecting a device to change addresses, so it would be necessary to retain the same BD_ADDR while paired, which does not provide much additional privacy so long as the device is periodically discoverable. A good understanding of the role of the LAP, NAP, and UAP [2B, 6.5.1.4] and [2B, 1.2], and what can be changed is important if attempting to change the device address.]

Why should I do this?

While a BR/EDR is discoverable, its BD_ADDR can be discovered by any other device within range, and passively when the BR/EDR device responds. In both cases, the BD_ADDR is immediately known, and this can be used as a permanent identifier to track the device. Where a dual-mode device (supporting both BR/EDR and LE) is in use, providing discoverability via BR/EDR can invalidate any privacy gains from using an LE privacy mode, so if a Bluetooth device must be made discoverable via BR/EDR, it should be for as short a duration as possible, or only during connection establishment.

What are the implications of doing this?

A device that is only discoverable while pairing, or while in a connectable mode is not conveniently discoverable. A user of this device must know that it must be placed into a discoverable mode and must know how to do this. While discoverable, the device will still be trackable, so the very act of enabling device discovery in a public setting may immediately permit a known BD_ADDR to be found by an attacker (e.g. who has seen or interacted with the device in the past or who has access to a BD_ADDR database culled from other interactions). Pairing modes are a common enough idiom that requiring discoverability only during pairing may be sufficient to reduce the privacy attack footprint.

When should I do this?

For some devices, remaining discoverable while Bluetooth technology is enabled is a logical design choice, but it has privacy implications. Providing a user the ability to disable discoverability independently represents a good compromise but may result in user confusion. For peripherals, remaining discoverable only while in a pairing mode is often the best approach, because being non-discoverable does not imply being non-connectable.

Why might I not do this?

Where a device is designed to be readily and easily discovered and connected to by various disparate devices at all times, or where a device is in a fixed location and not subject to identification with a particular user (e.g. a camera), being discoverable all the time may have security implications, but may not represent a privacy concern. In such cases, a device may reasonably remain in general-discoverable mode at all times.

[NOTE: For dual-mode devices, using LE in privacy mode with a private address to initiate a connection and using an encrypted connection to exchange a fixed BD_ADDR provides another method for improving the privacy footprint of a device supporting BR/EDR. There exists no standard method for identifying the BR/EDR address from the LE side, however pairing keys can be shared between LE and BR/EDR on a dual-mode controller using cross-transport key derivation [3C, 14.1], so it is not necessary for a user to independently pair both LE and BR/EDR connections].

5.5 [BR-EDR-5] Reject new baseband links when the maximum number of devices supported are already connected.

When a device only supports a single remote device (e.g. a HF device), or a small set of connections, the number of active connected devices is necessarily smaller than the number supported in a piconet (noting that PARK mode is deprecated). A device that is non-connectable [3C, 4.2] has a smaller attack footprint, but also requires interaction to place it explicitly into a connectable mode, which is not generally desirable from a user experience perspective. A good balance for a device that is normally connectable is to accept new connections only when it does not have all of the baseband links it supports already established.

Why should I do this?

If a device remains connectable even when it has established all of the baseband links it is intended to support, it can leave open any attack vectors, from DoS attacks from a device connecting with an address matching an already connected device to an attempt to gain information about the bits in a key (legacy pairing) by repeatedly triggering secure authentication with a deliberately non-random LMP_au_rand [4]. Any current or future attack vector that exploits weaknesses in BR/EDR pairing, authentication, or pairing can only be exploited when a device is establish a new baseband link. Therefore, it makes sense for a device that is actively connected to all of the devices it supports simultaneously to disallow further baseband links to other devices until one or more of the existing baseband links has closed.

What are the implications of doing this?

Under normal circumstances, limiting new baseband links when there are no more connections permitted has no impact on the behavior of the device from a user perspective other than eliminating certain attack vectors. There exists a risk of a DoS attacker parking on a security mode 1 connection (e.g. an SDP query) to halt other devices from connecting. The choice can be made to only limit connections when service-level (or even encrypted) links are established with the set number of supported devices.

When should I do this?

As a general rule, disabling discoverability and/or connectability when all permitted baseband links are established is good practice. A device that does not do this is not necessarily more vulnerable than one that does, but if any future attacks on a device are discovered, including implementation-specific issues on any unsecured profile, limiting when a device permits new connections decreases the likelihood that a user implementation is affected, but can also limit the establishment of new L2CAP connections from an existing device. In such cases, connections from an existing device may be permitted, but others rejected.

Why might I not do this?

If a device is intended to be always connectable and support the maximum number of devices in a piconet or if it is intended to permit service discovery even when connections are already established to it, it may be undesirable to ever make the device non-connectable (except for an airplane-mode or equivalent). The conditions under which a device many want to remain connectable or permit new baseband links are implementation dependent.

5.6 [BR-EDR-6] Do not store link keys in the controller.

The core specification recommends that devices using secure connections only mode should not store link keys in the controller [4E, 7.3.9].

Why should I do this?

Ensuring that link keys (and other secure data) are not present on the controller ensures that the controller cannot use the link key to establish legacy authentication with a known link key. This is essential to avoid an attacker attempting to downgrade the security of a link by attempting a legacy authentication as a previously paired device. Even when the authentication cannot complete, this still offers vectors for an attacker attempting to gain information about the key through repeated trials.

To a lesser extent, this recommendation exists to firewall the link keys from the controller when not actively being used for authentication or encryption. Though Bluetooth controllers are generally secure from direct access to their internal RAM/ROM image from a remote attacker, Bluetooth controllers can also be difficult or impossible (if firmware is entirely in ROM) to patch if an exploit is found that affects them. Secure connections only mode is intended to be FIPS compliant. For implementations that desire a higher security level [8], controlling the storage of the link key becomes an important consideration, which is not possible if the controller is storing it.

What are the implications of doing this?

Storing the link key on the host requires that storage space (persistent (e.g. EEPROM) or semi-persistent (e.g. battery-backed RAM)) be reserved to record the link keys and the associated Bluetooth addresses. It also requires that the software provide the link keys to the controller upon request. This may present an opportunity to maintain the link keys in a secure (e.g. encrypted) local storage on the device, which can reduce the opportunity for a remote attacker (e.g. on a smartphone) to acquire the link key remotely.

When should I do this?

This recommendation should be followed when Secure Connections Only Mode is used and space exists on the host to store the link keys for bonded remote devices.

Why might I not do this?

If a host does not support a permanent or semi-permanent place to store link keys, it may not be possible to retain the link keys in the host as recommended. It is still possible to use the controller to store these link keys on behalf of the host.

6 Bluetooth® LE and BR/EDR Coexistence Recommendations and Best Practices

6.1 [LE+BR/EDR-1] Ensure that LE and BR/EDR authentication methods are equally strong when supporting cross-transport key derivation and sharing keys between transports.

For devices that support both BR/EDR and LE connections simultaneously, cross-transport key derivation permits those devices to derive the link key (BR/EDR) from the LTK (LE) and vice versa [3H, 2.3.5.7] under certain conditions, avoiding two separate pairings. The specification requires that a key already generated not overwrite a key of lesser strength or less MITM protection (e.g. a non-authenticated key must not overwrite an authenticated key) [3C, 14.1]. It is important to ensure that these limitations are not bypassed by a local key database when storing an LTK or link key, or when responding to a query for an existing LTK or link key for LE or BR/EDR (respectively). It is also desirable to use the most secure (including privacy considerations) available method to generate a key when both BR/EDR and LE transports are both supported.

Why should I do this?

In general, because the LTK in LE and the link key in BR/EDR are derived by similar methods and have the same basic length and the same purpose (a symmetric key used for the derivation of encryption keys), the two keys are effectively interchangeable. When a dual-mode controller (or a device with multiple Bluetooth radios) can generate an LTK or a link key by different methods, the security of the method affects the use of the key. If the key was obtained by a Just Works BR/EDR pairing, it cannot be used to derive a key providing access to a GATT attribute that requires authentication. Conversely, if an LTK was generated using an insecure protocol (e.g. LE legacy pairing using PassKey), using it to derive a key to secure a BR/EDR link (e.g. a HID keyboard) would expose that link to the same attack vectors and so is not permitted [3H, 2.3.5.7].

To ensure that a downgrade of security never occurs as a result of using cross-transport key derivation, it is essential that a host track not only keys used but also the method used to generate the key. It is also important to store both keys securely (if they are not derived as needed). This means that an LTK generated by a LE Secure Connections pairing should never be stored in the controller as a BR/EDR link key.

What are the implications of doing this?

Ensuring that a key derivation from another transport does not overwrite an existing key derived using a more secure method (e.g. authenticated vs. non-authenticated pairing), ensures that the security limitations of one transport’s pairing procedure cannot carry over to the other transport and result in an insecure link where a secure link was previously available. It requires that the host be aware of the method used to generate a key and how secure the original key is (e.g. whether the public key used in the key exchange procedure was valid for the elliptic curve used).

When should I do this?

The host should never use a key to encrypt a link when it would result in a downgrade in the required level of security of the link. Using a key derived from another transport indiscriminately, even if such a key assignment is not downgrading the security of an existing key, risks the misuse of a key that was provided by a MITM, or was downgraded by attacker performing an injection attack during an ECDH public key exchange [13]. Care should always be taken to ensure that when a key is transformed from a LE LTK into a BR/EDR link key, information about whether the key was derived from a secure and authenticated pairing is not lost.

Why might I not do this?

Protection against a MITM gaining access to an otherwise secure profile through an alternate transport should not be bypassed. A host should never indiscriminately use an LTK as a link-key or a link-key as an LTK without tracking how the keys were derived.

6.2 [LE+BR/EDR-2] Treat devices with both LE and BR/EDR transports that use Cross Transport Key Derivation as providing access to the same security domain through each transport.

Even when care is taken to ensure that a key with lower encryption key length or generated without MITM protection does not overwrite a key with greater strength or protection level using CTKD [LE+BR/EDR-1], a remote peer establishing encryption using an association model that provides no MITM protections or which can be readily compromised, still may provide access to resources on both transports in a manner that permits an attacker access to private information or resources that an end-user may not be aware of or in control of. It is important when CTKD has been used to establish keys on both transports, to always treat both transports as representing the same security domain.

Why should I do this?

When CTKD is employed to automatically translate a link key to an LTK or LTK to a link key, this makes the security domain being accessed from one transport functionally equivalent to the security domain of the other. With some specific (and explicit) restrictions, each key provides the same level of access as the other, and there is not an effective difference between pairing with LE and pairing with BR/EDR. Doing either pairing permits access to both transports with what are effectively the same credentials with the same level of protection.

With this security-domain equivalence comes a responsibility to treat these two transports as providing equivalent access to a peer that can access both. If an LE service that provides access to phone-book data requires MITM protection to access attributes that present a privacy risk to the user, but equivalent access is available through BR/EDR to a user of a phone-book access profile (PBAP) without requiring MITM protection, then the LE service's increased level of protection is bypassed by the equivalence of the security domain between BR/EDR and LE transports. Even though the LE service may not permit a non-MITM-protected key derived from the unauthenticated BR/EDR key to access the user's private data, the access through BR/EDR is sufficient to acquire or manipulate the same data.

Additionally, the ability of a transport to negotiate the key strength during encryption establishment means that each time a baseband link is established, and encryption is enabled, the effective level of protection from using that link may differ (generally it should not change, but it can). If a service is reasonably protected by a link using 128-bit encryption keys but not by one offering 56-bit, it's also important to restrict access to both LE and BR/EDR services offering the same access when 56-bit key strength is used, if 56-key strength is considered insufficient for either transport.

As such, it is imperative that implementations that protect user data understand the limitations of CTKD when it is used and place the same restrictions on access to data regardless of the transport used to access it. Equivalently, if there are some services/profiles accessible through LE that are not available through BR/EDR (or vice versa), then those services effectively are accessible through both transports, at least from the perspective of any intended restrictions on access.

What are the implications of doing this?

Ensuring that access to services/profiles from one transport does not offer more access than afforded to users accessing the same service/profile or same information from the other transport ensures that attackers cannot leverage CTKD to gain undesired access. In particular, JustWorks authentication does not afford protection against MITM attacks, and allowing a JustWorks-paired peer to access resources requiring MITM protection on the opposing transport, is equivalent to not requiring MITM protection for those resources.

When should I do this?

Whenever user resources or device behaviors are protected against access or modification by restricting access to those resources to encrypted, authenticated, or authorized links or where access to resources are restricted to links with specific encryption key length minimums or encryption mechanisms, the same or similar resources on the other transport need to have equivalent restrictions if CTKD is used.

Why might I not do this?

For devices that are LE only or BR/EDR only, CTKD is not used. Otherwise, implementations should always be aware of what is accessible based on the key strength, encryption mechanism and MITM protection or authorization requirements and ensure that LE and BR/EDR are treated as offering the same access to the same security domain when CTKD is used.

6.3 [LE+BR/EDR-3] Ensure that MITM protection is enabled for access to a profile even when the profile specification permits reduced security.

Most profile specifications provide guidance on what security modes are recommended or required for profiles and service characteristics. For flexibility, most profiles permit multiple options for security, but some require specific Bluetooth security modes and levels or require encryption, authentication, or authorization for specific service characteristics. Because the nature of the data carried by many profiles is application-specific, actual risks to user security or privacy vary depending on what the data represents. In some cases, this is clearly delineated (biometric data, for example, will always represent a privacy risk if exposed), in other cases the effects of exposure are less clear.

Why should I do this?

When data presented by a profile exposes a clear risk to a user from exposure to an attacker, including a MITM attacker, additional security measures are appropriate. The profile requirements may not include a mandate that the highest available level of security be used, because there may be reasonable arguments that certain uses of the profile will not require this level of security. The Bluetooth specifications provide flexibility in these cases, even where the risks could be high if the choice of minimum security-mode and level is not the most secure available. When MITM protection is optional, it is up to an implementation to make an informed decision as to whether MITM protection is appropriate or necessary, or whether additional authentication procedures are warranted (e.g. authorization in Bluetooth LE or additional vendor-specific security measures).

What are the implications of doing this?

For Bluetooth BR/EDR and LE implementations that require MITM protection, it is necessary that the input/output hardware in use in both the Central role and Peripheral role devices support an available MITM-resistant pairing association model for supported MITM protection methods to be used. This includes Passkey and Numeric-Comparison procedures, or an OOB mechanism where the display or input capabilities of the pairing devices do not meet the requirements of the other association models and a common OOB mechanism is supported. Providing sufficient MITM protection ensures that the risk to an end-user from an eavesdropped, intercepted or manipulated pairing session is reduced or eliminated.

When should I do this?

As long a device's input/output hardware architecture permits the application of MITM protection methods for securing a profile carrying sensitive user or application-specific data, MITM protected modes should always be used. When an application requires stricter control over the security of data, use of an authorization method is also recommended and should be implemented, presuming that the peer devices can agree on a protocol.

In those cases where a device does not typically have support for one of the available mechanisms required to introduce MITM protections, it should be considered whether other mechanisms may be available on the device (e.g. voice prompts on a HF device) to permit MITM protections to be introduced.

Why might I not do this?

It is not always possible for a device to support Bluetooth MITM protection given the available authentication mechanisms. The input/output capabilities of the device may not allow for maximum security to be enabled. Generally, the maximum protection level supported by the input/output capabilities available in the hardware configuration should be used. The only advantage to less secure association models in Bluetooth pairing is their relative simplicity and potentially simplified hardware requirements. An association model without MITM protection may sometimes be sufficient if used in an appropriate venue (e.g. away from a public place where an eavesdropping or MITM attack may be present), so it is not always the case that a lack of MITM protection represents an immediate security risk, but where the data carried is of importance, the security of that data may depend on effective MITM protections, because even a location that seems safe from a MITM attacker could be compromised if the information being attacked is valuable enough.

6.4 [LE+BR/EDR-4] Provide profile-specific authorization/permission granularity around profiles and services that carry a user’s private data or offer control over device behaviors.

The Bluetooth baseband’s security for peer-to-peer communications treats the baseband link as the sole resource requiring protection. The default Bluetooth architecture does not include a mechanism for securing against a threat imposed by other users or processes that run on a device, nor does a Bluetooth link provide intrinsic restrictions on what services the peer may access depending on its security role, as there is no security role defined. This means that if a peer can access a device’s resources with a particular Bluetooth security mode and level, it can access any resources accessible with that mode and level.

This behavior may offer sufficient security, especially if the peer device is owned and operated by the same user that owns and operates the Bluetooth Platform device with which it is bonded. It some cases, the user does not own the peer device. For example, a room-lock in a hotel may use pairing temporarily to ensure integrity of an unlock request from an App that provides room access. This does not imply that the App or the lock should be able to read the user’s phone book, make a call on behalf of the user, or access a user’s location history (e.g. if the lock uses the recommended Security Mode 1 Level 4). Some profiles (e.g. PBAP) require explicit user authorization to access private user data, others do not. It may be valuable to treat a paired/bonded peer as potential attacker and ensure that a user’s private data is protected by additional mechanisms to those offered by the Bluetooth specifications.

Why should I do this?

Bluetooth security is primarily focused on limiting the ability of a third-party to gain access to or control over the data carried by the baseband link between a device and its paired peer(s). A peer device is offered a set of profiles and services by the device, and without additional security measures, all the services offered that are accessible at the security mode and level of the pairing with the peer are normally accessible to that peer.

The various security modes and levels offered by the Bluetooth security procedures provide different levels of protection against attack by a third-party. Authorization in LE can ensure that a peer’s access is authorized by the device or the user of the device, but it still provides full access to all the device attributes protected by authorization once authorization has been granted, regardless of the purpose of the peer device, or who controls it.

If the device’s profile and services offer different behaviors with different effects, different scopes, or different levels of disclosure of potentially private data, the single layer of access restriction offered by the Bluetooth specifications may be insufficient. A public peer that you wish to communicate with securely will gain access to services on your local device that it may use to acquire information about you.

If you want to stop a device from permitting access to all profiles accessible at a given security mode and level, additional access controls are needed.

What are the implications of doing this?

If the device and its peer are required to negotiate additional granular profile-access permissions, both must offer a common security protocol that may not interoperate with all devices that a user may wish to use. This compatibility problem can be resolved, in part, by limiting the authorization restrictions to the local device, permitting a user to explicitly authorize or deny access to data by a peer. However, a user may not be able to similarly enable similar limits for a Peripheral with limited or no user interface.

Such granular authorization also assumes that a user has knowledge of what data a user may access through a profile, or when the permissions relate to a more generic requirement, that a profile implementation is able to properly restrict access to resources that may be available through vendor-specific means (e.g. vendor-specific HFP AT commands that permit access to SIM-card data).

When should I do this?

Whenever a profile offers up private information about a user, and that profile is potentially accessible to a third-party peer device that the user has paired with, it is important to provide some protection against malicious acquisition of that private data by that peer device, otherwise a seemingly innocuous pairing may be used to clandestinely acquire information that should be secured from access. Even where explicit authorization is required for a peer device to access certain local profiles (e.g. in a medical device), it may not be appropriate for that peer to also access all other profiles requiring authorization.

Why might I not do this?

There is a clear burden on a user if every access of a peer device requires authorization of access or updates to permissions, especially if the peer device is clearly intended to have that access. There is a need for balance in selecting which profiles or services may require additional levels of confirmation before a peer may access the access-controlled resources. Users can become conditioned to respond in the positive to requests for access if a sufficient number of requests for access authorization are posed. It is also important to avoid superfluous requests for authorization, especially a simple yes/no question, to avoid the natural tendency to accept or reject all access indiscriminately. It is also important to treat erasure of the link key or LTK between peers as a condition that resets the permissions of that device (if permissions are granular per device), otherwise a spoofed address may be used to gain the permissions already granted to another device, especially where a JustWorks association model is permitted.

If a peer requires access to a certain minimum set of profiles to perform its normal actions and a user does not provide sufficient authorization, the behavior of a peer device may be hobbled but without a means of identifying to the user that the continued restricted access is the cause. It is important to provide enough access to permit the intended function of a group of devices to occur (e.g. if HFP access or AVRCP access requires permissions these may be introduced along with HFP, AVRCP, and A2DP).

7 Bluetooth® Profiles Recommendations and Best Practices

7.1 Medical-Device Profiles and MITM/Privacy

Devices that support acquisition or access to biometric data may present significant privacy risks to their users. Those profiles that also provide direct control of medical devices (e.g. pacemakers or insulin pumps) need even more careful control of device security to ensure that risks to health or life are not exposed. It is beyond the scope of this guide to identify all of the requirements for safe control of medical devices, however implementations choosing to define a custom profile for a medical device should take care that minimum safeguards are in place:

Require authorization.

The insulin delivery service requires the authorization permission for all characteristics, this is done to ensure that only an authorized party (e.g. the device owner) may access the attributes in the profile. It is recommended that even where not explicitly required, devices carrying user medical information require authorization on attributes related to the primary biometric functions of the device.

Enable Privacy.

With medical profiles or devices that provide access to biometric measurements, the privacy of the user may be an important consideration. The presence, for example, of a blood pressure profile (BLP) on a device, where that device is not a general-purpose peripheral, provides medical details about the device's user, namely that they use a blood-pressure monitor.

Do Not Include Advertisement Data Payloads when Bonded.

Many medical profiles already recommend that devices not include certain advertising payload data when in privacy mode. (**[LE-10] 4.10**) recommends that advertisements with data payloads not be generated by devices using private addressing modes unless the data is obfuscated to avoid tracking by an attacker.

7.2 Location Services and Privacy

Although location services like the location and navigation profile (LNP) and indoor positioning service (IPS)) do not intrinsically provide privacy risks just by their presence, access to these profiles, if not authenticated (e.g. by allowing JustWorks pairing), may provide immediate access to the precise device location.

Device privacy is not required with the LNP and IPS services, but its use is recommended to avoid associating a specific device with a location service.

7.3 OBEX-Based Profiles and File Tree Access

Classic BR/EDR supports a number of profiles based on the object exchange (OBEX) protocol. Object queries in OBEX-based profiles are based on a query of an object in a tree (for example file transfer profile (FTP)). There are a number of classic BR/EDR vulnerabilities where attackers use the access to other OBEX-based profiles to gain information from the object tree that is not intended.

For example, care must be taken when accessing files, contacts, or phone book entries in OBEX, that general queries from a remote device are not passed verbatim to a file-system, especially a file system that supports moving up a tree hierarchy using inline escape codes [e.g. '..']. In the same sense that web services must not permit arbitrary access to artifacts in the web server's file system, OBEX services must similarly restrict access to just those artifacts that the remote device is meant to access. For example, a PBAP profile may place vCARD entries in a real file folder, so that access to /telecom/pb/0.vcf could be through a real location like '/home/User/.bt/pbap/telecom/pb/0.vcf'. Even if SETPATH does not support the backup flag above the root at the '/home/User/.bt/pbap/' folder, a path string containing '..' (e.g. telecom/../../../) might result in access to an undesired location if handled naively.

7.4 HFP and Pin Codes

For the purposes of providing backward compatibility or simplicity of pairing procedures, most HFP devices support pairing with a fixed pin code. Though common HF devices require being placed into a pairing mode before pairing with a new remote device, the use of legacy pairing, if captured by an eavesdropper, does not afford protection against an attack on the link key. If legacy pairing is used, pairing in a public space introduces the risk of subsequent encrypted links (e.g. SCO or eSCO audio links) being decoded by an attacker in possession of a capture of the pairing procedure.

Since pin codes for HF devices tend to be fixed and well-known, they do not provide any protection for the end-user. It is recommended that devices support and prefer secure simple pairing, and support authentication (e.g. through limited text-to-speech).

7.5 HFP and AT Commands

7.5.1 [HFP-AT-1] Filter unknown or unintended AT commands.

The hands-free profile is based on the serial port profile implementation. The majority of data transferred, except for the audio connection, are commands and events based on a subset of the Hayes modem commands in common use with GSM WWAN (e.g. cellular) radios [10]. Because the majority of HF-role implementations feed commands not explicitly managed by an AG profile directly from a remote HF role to a WWAN radio module, HF devices often send commands not specified by the HFP profile specification in order to trigger custom behaviors or to enable special features (e.g. Apple Siri support). An attacker gaining access to a phone through a local AG profile can send commands which may compromise information about the phone's owner through custom AT commands or potentially initiate SMS messaging or calls. Ensuring that authentication is used with HFP can reduce the availability of the HFP profile to attackers, but some care also should be taken to limit what commands an HF-role device may interpret.

7.6 SAP and Profile-Based Encryption Key Sizes

The encryption key size negotiation process results in a key-size agreement between devices establishing encryption in BR/EDR. Certain profiles explicitly restrict the length of the encryption key used to ensure that a connection used to carry profile-specific data cannot be eavesdropped by an attacker attempting a brute force on the encryption key. An example of this requirement is the SIM Access Profile (SAP), which requires both that the maximum key length is supported and that the minimum key length be no less than 64 bits. Other profiles provide similar restrictions as recommendations, but rarely are the lengths explicitly enforced.

Each service-level connection (the L2CAP connection) may recommend or require a specific encryption-key size. Unfortunately, the baseband link is what controls this size of the encryption-key, and once the baseband link encryption is established, the key size has already been chosen. If a profile needs to establish a connection over an existing baseband link and the encryption-key size that was already chosen is too small to properly secure the data the new profile connection exchanges, there does not exist a common procedure to fix this. In most cases, the new connection cannot be established without tearing down the encrypted link and re-establishing with a greater minimum encryption key length.

7.7 PAN/BNEP

The classic BR/EDR personal area network (PAN) profile is a lightweight wrapper around ethernet frames carried by the Bluetooth network encapsulation protocol (BNEP). Because PAN doesn't provide built-in authorization for a device connecting to a network through a Bluetooth link, it represents a vector for attacks on a network where the Bluetooth PAN profile's layer 2 protocol may exist behind a firewall that may otherwise protect against IP intrusions.

If possible, PAN servers should provide a NAT (Network Address Translation) to bridge a Bluetooth specific subnet to any local subnet on which a PAN-enabled device resides. Connections or multicast/broadcasts to and from this Bluetooth specific subnet should have initially restrictive and configurable firewall rules to avoid remote devices gaining access to a local network through physical access to a local PANAP (access point) or PANG (group ad-hoc) service. Treating a remote PAN device as part of a DMZ restricting access to local network resources is recommended unless an authorization mechanism is used to identify specifically authorized devices.

8 Common Misperceptions

Reading through the Bluetooth specifications can be a daunting undertaking with some unintended pitfalls. For those who wish to understand the deeper meaning of specific nomenclature, it's not always possible to use the most obvious interpretation afforded by a literal reading. This is frequently due to the need to retain the same nomenclature describing features even as those features become obsolete. For example, *LE Secure Connections Mode*, described in [1A, 5.1] sounds like the best possible mode for ensuring maximal protection of confidentiality and integrity of data carried between LE-supporting devices. Arguably the more secure option is *LE Security Mode 1, Level 4* (a subset) as it demands mechanisms for protection against MITM attacks during pairing [3C, 10.2.1]. This fact can be derived from a careful reading of the specification, but it is by no means obvious. The following section enumerates some of these implicit assumptions to help correct easy misconceptions.

8.1 [G-1] Secure Connections Only Mode (LE), LE Secure Connections and LE Security Mode 1 Level 4

Secure Connections Only Mode requires the use of LE Security Mode 1 Level 4 [3C, 10.2.4] for any services that support it (a service may support only Security Mode 1 Level 1 and still be permitted). A

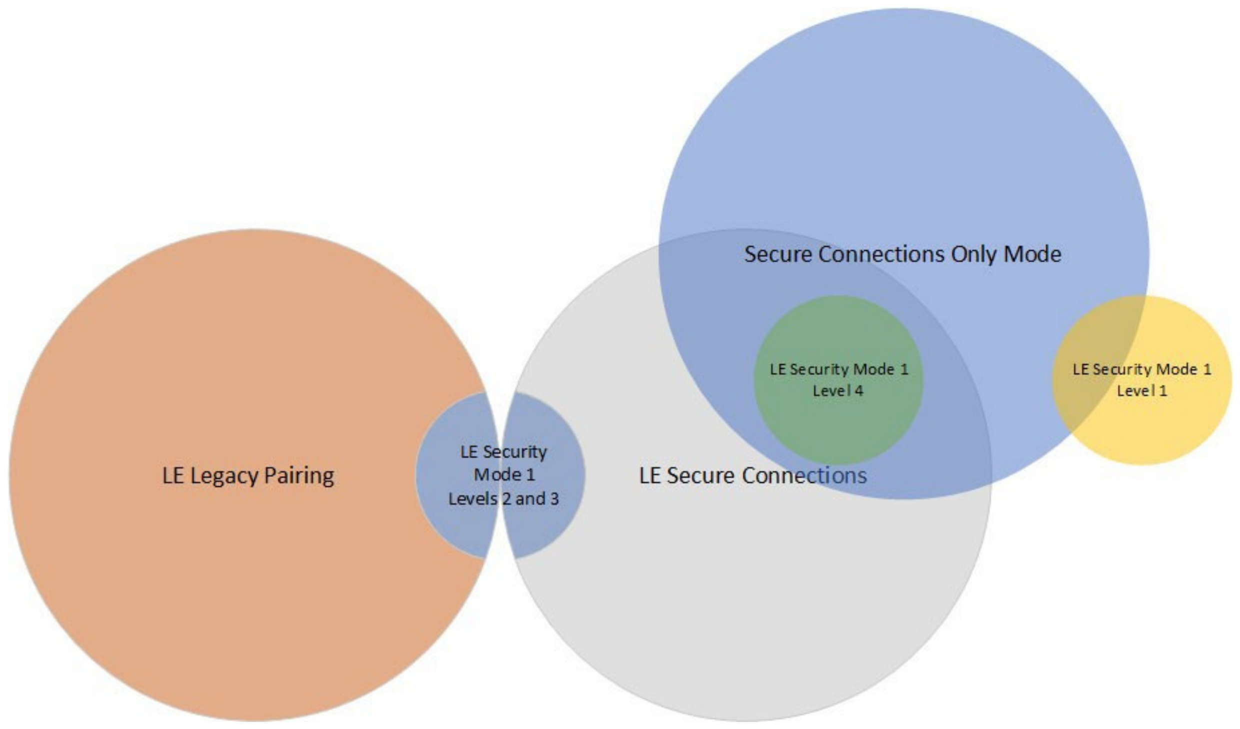
Secure Connections Only Mode
LE Security Mode 1 Level 4
LE Security Mode 1 Level 1
LE Legacy Pairing
LE Security Mode 1 Levels 2 and 3
LE Secure Connections

Figure 7 -- LE Security and Pairing Modes (Venn Diagram)

device not in Secure Connections Only Mode may still use and prefer Security Mode 1 Level 4, but it is not required to use it. LE Secure Connections is not the same as Secure Connections Only Mode but rather refers to the pairing and encryption mechanisms that are used when establishing a pairing or bonding relationship with a device and establishing an encrypted link. Contrast LE Secure Connections with LE Legacy Pairing, which is the legacy version of the same set of features.

8.2 [G-2] Secure Connections Only Mode is not Set-And-Forget

It is recommended that devices that do not need to use legacy pairing modes and that carry sensitive user information restrict themselves to Secure Connection Only Mode to disallow insecure pairing and encryption. There does not exist a flag that sets a controller to Secure Connections Only Mode, however. The requirements for the mode are defined in various portions of the specification and require the host (and possibly application) to act to ensure that legacy connections are not permitted while in this mode.

The following recommendations and restrictions are described in the specification for maintaining a device in Secure Connections Only mode:

- Enable Secure Simple Pairing Mode [4E, 7.3.59].
- Enable secure connections host support (usually a host stack behavior) [4E, 6.39] and [4E, 7.3.92].
- Refuse incoming and outgoing service-level connections from/to devices indicating no support for secure connections (host or controller support) in the feature mask [3C, 5.2.2].
- Refuse incoming and outgoing service-level connections from/to devices requesting unauthenticated pairing [3C, 5.2.2].
- Refuse incoming connections requiring Security Mode 4 Level 4 when that security mode and level is not supported by the remote device [3C, 5.2.2].
- Require encryption key size to be no less than 16 octets (host) [3C, 5.2.2.8].
- Do not store the link key in the controller [4E, 7.3.9].

8.3 [G-3] Just Works is only secure if nobody is around while you're pairing/bonding.

The Just Works pairing mechanism is provided in BR/EDR Secure Simple Pairing [1A, 5.2.4.2], LE Legacy Pairing [3H, 2.3.5.2], and LE Secure Connections [3H, C.2.2.2.1]. In general, despite the common name, these Just Works mechanisms are not identical (though BR/EDR secure simple pairing and LE secure connections use the same basic procedure). They are all vulnerable to straightforward attacks, though which attack can succeed depends on the flavor of Just Works used:

	BR/EDR Secure Simple Pairing Just Works	LE Legacy Pairing Just Works	LE Secure Connections Just Works
Eavesdropper seeing pairing can decode link (and possibly inject packets)	NO	YES	NO

Eavesdropper that saw bonding can decode subsequent connections (and possibly inject packets)	NO	YES	NO
MITM spoofing both devices during pairing can decrypt and inject packets	YES	YES	YES
MITM that spoofed both devices during bonding can decrypt and inject packets on subsequent connections	YES	YES	YES

Table 1 -- Attack Flavors on JustWorks

For both BR/EDR Secure Simple Pairing and LE Secure Connections, an attacker can only decrypt an encrypted link if the attacker succeeds in a MITM attack (attacks on the Diffie Hellman key exchange itself are considered to be impractical for sufficient key lengths [11]). An attacker with a recording of either the BR/EDR Secure Simple Pairing or LE Secure Connections JustWorks procedures does not have knowledge of the private key of either device, and without that information, the key exchange is effectively secure. This does not stop a MITM that spoofs both devices during pairing from establishing a pairing with each of the attacked devices. With Just Works, the numeric comparison values each device computes won't match (except with a 0.0001% chance), but because the devices don't reach out to the user to confirm that the values matched (presumably because they are unable to display or enter those numbers), the attacker can proceed without alerting the user to their presence.

In contrast to the Secure Simple Pairing and LE Secure Connections scenarios, the LE Legacy Pairing Justworks uses a temporary key (TK) value with a known value when pairing using JustWorks. Because this value is known, the computation of the short-term key (STK) can be made by an eavesdropper with just the information transmitted over-the-air, as long as the eavesdropper is present for the pairing procedure.

In all cases, once the key exchange has completed successfully with no intervention by a MITM or recording by an eavesdropper, the link key or LTK exchanged is no longer at risk of exposure except by brute-force attack, by information leakage (e.g. an attacker gaming BR/EDR authentication through multiple trials [2H, 5]) or future weakness found in the encryption algorithm. In practice, this means that LE Legacy Pairing Just Works should always be avoided, and that Just Works pairing is only secure when it results in a bonding (i.e. pairing does not proceed each time) and that this occurred out of range of any possible attacker. Devices that must use JustWorks should recommend that pairing not be attempted in public venues.

8.4 [G-4] LE Legacy Pairing does not have forward secrecy.

Forward secrecy (sometimes perfect forward secrecy) is a property of a key exchange protocol where the compromise of a local private key used in the exchange of the shared key (not the same as the private key) does not lead to a compromise of the shared key [12]. In general, forward secrecy is a desirable property whenever the same private key is used to exchange multiple secrets. A compromise of the private key does not lead to a compromise of all of the secrets exchanged using that key, since each key exchange would need to be compromised independently (for Diffie-Hellman key exchange, the problem is that of solving a Discrete Logarithm problem over a cyclic group).

LE Legacy Pairing does not use a public/private key pair, but the properties of forward secrecy are not met for similar reasons. In the case of legacy pairing, each pairing attempt uses just one 'private' key, referred to as the temporary key (TK), which is 128 bits, but not all of the bits are always significant. This key is zero for Just Works [3H, 2.3.5.2], a 20-bit number for PassKey [3H, 2.3.5.3], and a 128-bit number for OOB pairing [3H, 2.3.5.4]. In the case of a fully random 128-bit TK shared OOB, an attacker would need to try 2^{128} possibilities (2^{127} on average) to compromise the security of the exchanged long-term key LTK, but for passkey that number drops to (2^{20} - 48,576) = 1,000,000 and for Just Works, that number is just 1. In each case, once the key exchange is compromised (given a recording of the pairing), that link is compromised, as are any other keys shared (i.e. IRK and CSRK).

For this, and other reasons, LE Legacy Pairing should be avoided when LE Secure Connections is supported (**[LE-1], [LE-4], and [LE-5]**).

8.5 [G-5] LE Security Mode 2 and CSRK

The Connection Security Resolving Key (CSRK) if supported by and exchanged between two LE devices may be used to sign data in LE Security Mode 2 to ensure that the data is not modified on transmission. This allows you to send data without encrypting and ensure that the data is not modified while you are transmitting it. When a device's state doesn't require protection against eavesdropping, but the integrity of the data is essential, this mode is sufficient and good for devices with limited power requirements that might not want to have the overhead of AES-CCM.

However, security mode 2 has flaws that may permit an attacker to use a certain class of replay attacks to get a node to change state in an undesirable way. See **[LE-8]** for a more detailed discussion. Since LE Security Mode 2 doesn't protect against certain types of replay attacks and permits an eavesdropper to still see all of the communications between devices, it is flawed. The same flaws are not present in Security Mode 1 Levels 2, 3, and 4, and the only added complexity in using these modes is the overhead of establishing an encrypted link and the per-packet cost of encrypting each packet using AES-CCM (AES-128).

9 Implementation Issues: Bluetooth® Applications and OS Integration

Bluetooth implementations tend to consist of three (major) distinct functional blocks. The controller, the host stack, and the application that drives the host stack. In Bluetooth nomenclature, the latter two components are all part of the host, but usually, an implementation uses an existing Bluetooth stack and profiles to perform the low-level Bluetooth protocol procedures and implements high-level behaviors through an API. Because the high-level application may have varying responsibilities depending on where the boundaries of the host stack API lies, applications may have different responsibilities on different platforms. Because of this, not all of the issues described in this section are applicable to every application. On a platform where the operating system handles Bluetooth authentication, applications do not have the responsibility to manage authentication. On many embedded platforms, the application will need to manage authentication. For those platforms where the platform subsumes these behaviors, these issues may apply to the operating system or the operating system interface, rather than the application.

IO Capabilities and Supporting LE Pairing

As described in [LE-15], it's important to correctly report the IO capabilities of a device when indicating what pairing methods are supported by a platform. The supported input and output capabilities [3H, 2.3.2] of both devices determine which authentication method will be used [3H, 2.3.5.1].

Because the input and output capabilities of a remote device are not known until devices initiate a pairing/bonding procedure, and various devices will have varying capabilities, it will be necessary to support more than one pairing method if maximum interoperability is required. If authentication is a requirement for a specific link, it may not be possible to pair with certain peripheral devices if those devices don't have some output capabilities or use a supported out-of-band mechanism. In such cases, downgrading pairing to Just Works just to establish a link is not recommended.

If a device supports multiple profiles and established an encrypted link as a result of accessing an attribute that does not require authentication, an encrypted link may be established with a MITM rather than the intended device. If the LTK is stored (bonding) and used for a subsequent link establishment resulting from access to an attribute that requires authentication, it's important that the stored LTK not be used for a link permitting access to the authenticated attribute, otherwise, a MITM can use a disconnect/reconnect sequence to achieve an escalation of access. Ensuring this restriction is applied correctly requires that the application (or OS) be aware of the level of authentication with which the LTK was exchanged previously. The specification does not currently address this case directly [3C, 10.3].

IO Capabilities and Supporting BR/EDR Secure-Simple Pairing

As described in [LE-15], it's important to correctly report the IO capabilities of a device when indicating what pairing methods are supported by a platform. The supported input and output capabilities [3C, 5.2.2.5] of both devices determine which authentication method will be used [3C, 5.2.2.6].

Because the input and output capabilities of a remote device are not known until devices initiate a pairing/bonding procedure, and various devices will have varying capabilities, it will be necessary to support more than one pairing method if maximum interoperability is required. If authentication is desirable for a specific profile, it may not be possible to pair with certain peripheral devices if those devices don't have some output capabilities or use a supported out-of-band mechanism. In such cases, downgrading pairing to Just Works just to establish a link is not recommended.

If a device supports multiple profiles and established an encrypted link for profiles that do not require authentication, an encrypted link may be established with a MITM rather than the intended device. If the link key is stored (bonding) and used for a subsequent link establishment resulting from a service level connection requiring authentication, it's important that the stored link key not be used for the service-level connection, otherwise, a MITM can use a disconnect and reconnect to achieve an escalation of access. Ensuring this restriction is applied correctly requires that the application (or OS) be aware of the level of authentication with which the link key was exchanged previously.

10 Acronyms and Abbreviations

Acronym/Abbreviation	Meaning
AES	Advanced Encryption Standard
AES-CMAC	Advanced Encryption Standard Cipher-based Message Authentication Code
AFH	Adaptive Frequency Hopping
AMP	Alternate MAC PHY
ATT	Attribute Protocol
BIG	Broadcast Isochronous Group
BR	Basic Rate
CI	Connection Interval
CMAC	Cipher-based Message Authentication Code
CRC	Cyclic Redundancy Code
CSPRNG	Cryptographically Secure Pseudo-Random Number Generator
CSRK	Connection Signature Resolving Key
E0	Bluetooth Encryption Cipher E0
ECB	Electronic Code Book (block cipher mode)
ECDH	Elliptic Curve Diffie-Hellman
EDR	Enhanced Data Rate
EDIV	Encryption Diversifier

Acronym/Abbreviation	Meaning
EEPROM	Electrically Erasable Programmable Read Only Memory
FHSS	Frequency Hopping Spread Spectrum
FIPS	Federal Information Processing Standards
GATT	Generic Attribute Profile
GHz	Gigahertz
GSM	Global System for Mobile communications
HID	Human Interface Device
HOGP	HID over GATT Profile
ISM	Industrial Scientific Medical
LE	Low Energy
LTK	Long Term Key
MIC	Message Integrity Check (c.f. Message Authentication Code)
MITM	Man in the Middle
NAT	Network Address Translation
OS	Operating System
PHY	Physical Layer
RADIUS	Remote Authentication Dial-In User Service
RAM	Random Access Memory

Acronym/Abbreviation	Meaning
RF	Radio Frequency
RFI	Radio Frequency Interference
SDP	Service Discovery Profile
RSSI	Received Signal Strength Indicator
WWAN	Wireless Wide-Area Network

Table 2: Acronyms and abbreviations

11 References

[1] https://www.bluetooth.com/bluetooth-resources/le-security-study-guide/

[2] IEEE Standard 802.15.1-2005 (withdrawn)

[3] FIPS PUB 186-4 Digital Signature Standard (DSS): (D.1.2.3, D.2.3)

https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf

[4] Ryan, Mike. *Bluetooth: With Low Energy comes Low Security.*

[5] https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-22r1a.pdf

[6] Matsumoto, Makoto and Nishimura, Takuji. *Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator.* ACM Transactions on Modeling and Computer Simulation 8(1): 3-30.

[7] Remote Authentication Dial-In User Service: https://tools.ietf.org/html/rfc2865

[8] FIPS PUB 140-2. Security requirements for cryptographic modules.
https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.140-2.pdf

[9] Kelsey, Schneier, Wagner. *Key Schedule Weaknesses in SAFER+.*
https://www.schneier.com/academic/paperfiles/paper-safer.pdf

[10] 3GPP TS 27.007:
https://www.etsi.org/deliver/etsi_ts/127000_127099/127007/13.03.00_60/ts_127007v130300p.pdf

[11] Commeine, Semaev. *An algorithm to Solve the Discrete Logarithm Problem with the Number Field Sieve.*[2]

[12] Menezes, Oorschot, and Vanstone. *Handbook of applied cryptography.* CRC Press, 1996.

[13] CVE-2019-9506. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-9506

[14] NIST Guide to Bluetooth Security : (NIST Special Publication 800-121 Revision 2)
https://www.nist.gov/publications/guide-bluetooth-security-1

[15] Diffie, Whitfield and Hellman, Martin E. *New Directions in Cryptography.* IEEE Transactions on Information Theory. 22 (6): 644-654.

[16] Winkler, Nolan. The Discrete Log Problem and Elliptic Curve Cryptography. 2014

[2] This approach posits a time-complexity for solution of the discrete logarithm problem of $L_P({}^1/_3, 3^{1/3})$ [L-notation], an improvement over prior methods. [$L_P(\alpha, c) = e^{(c+o(1))(\ln p)^{\alpha}(\ln\ln p)^{1-\alpha}}$ where P is the order of the multiplicative cyclic group, $0 \leq \alpha \leq 1, c \geq 0$, and $o(1)$ is a function $f(x)$ such that $\lim_{x\to\infty} f(x) = 0$]

[17] Jayant Bhoge, Prashant Chatur. The Avalanche Effect of AES Algorithm. International Journal of Computer Science and Information Technologies, Vol. 5 (3), 2014, 3101 – 3103.

[18] Genkin, Pachmanov, Pipman, Tromer and Yarom. *ECDSA Key Extraction for Mobile Devices via Nonintrusive Physical Side Channels.* ACM Conference on Computer and Communications Security CCS 2016.

[19] SEI CERT C Coding Standard: Rules for Developing Safe, Reliable, and Secure Systems. Carnegie Mellon University, 2016.

[20] SEI CERT C++ Coding Standard (2016 Edition). Carnegie Mellon University, 2017.

[21] Davis, Martin. Computability and Unsolvability (Reprint Edition). Dover Publications, 1982.