AN1302: *Bluetooth*® Low Energy Application Security Design Considerations in SDK v3.x and Higher

This application note provides details on designing Bluetooth Low Energy applications with security and privacy in mind.

KEY POINTS

- Overview of Bluetooth Low Energy security and privacy
- Using the security features built into the stack/SDK
- Implementing application layer encryption
- Secure boot

Copyright © 2023 by Silicon Laboratories

1 Introduction

The Bluetooth specification makes security and privacy features optional. While using these features is not required by the specification, it is highly recommended to take advantage of the highest security features available. The Bluetooth SIG has published the following Security and Privacy Best Practices Guide. This guide contains information for implementers of stacks as well as applications. Silicon Labs suggests following as many of these recommended best practices as possible.

1.1 Threat Models

Several threats exist when communicating between two parties:

- Passive eavesdropping - an unauthorized third-party intercepts sensitive data.
- Man-in-the-middle (MITM) - an unauthorized third party injects or modifies data.
- Denial-of-Service through Wi-Fi coexistence.
- Tracking - an unauthorized third party can track the location of a moveable device.
- Spoofing – a device mimics the identity of a trusted device.

1.2 Bluetooth Security Concepts

This section describes some basic concepts and terminology regarding Bluetooth security.

1.2.1 Connections, Pairing and Bonding

A connection is required for reliable data exchange between two Bluetooth Low Energy devices as well as encryption and authentication, which are optional.

Pairing refers to a one-time secure relationship between two devices to establish cryptographic keys and allows data to be exchanged securely for the life of the connection. Loss of the connection results in termination of the pairing.

Bonding refers to a persistent relationship where cryptographic keys are established and stored in non-volatile memory and can exist over multiple connections.

LE Secure Connections refers to a method for exchanging cryptographic keys using the elliptic curve Diffie-Helman (ECDH) technique.

1.2.2 Security Modes and Levels

Security mode 1 is the only mode supported for Bluetooth Low Energy in the Silicon Labs' stack. The levels are as follows:

- Level 1 - no security
- Level 2 - unauthenticated pairing with encryption
- Level 3 - authenticated pairing with encryption
- Level 4 - authenticated secure connections with strong encryption (ECDH key exchange)

1.2.3 GATT Database

Security in Bluetooth Low Energy is primarily controlled through GATT characteristics, which can have the following properties:

- Encrypted read/write/notify/indicate
- Authenticated read/write/notify/indicate
- Bonded read/write/notify/indicate

A GATT client attempting to access a characteristic must support the properties required by the characteristic.

2 Working with Stack Security Features

2.1 Securing Connections

Bluetooth Low Energy connections can be secured in one of two ways:

- The peer device initiates an encrypted connection
- The connection is explicitly secured through an API call

2.1.1 Protecting GATT characteristics

Silicon Labs provides a GATT configuration tool that allows users to create a GATT database from a graphical user interface shown in the following figure.

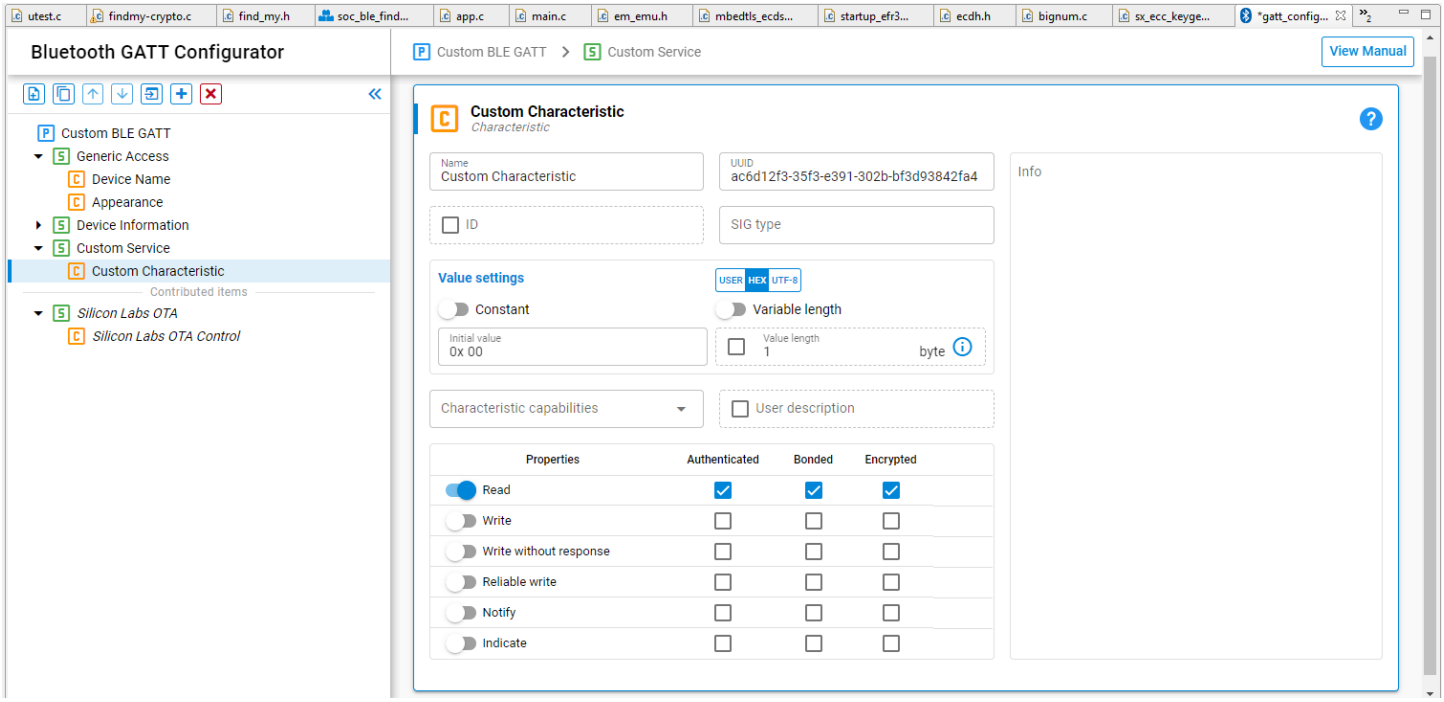


Figure 2-1. Bluetooth GATT Configurator

As shown, the properties such as read, write, notify, and so on can be enabled or disabled, and each property can be given the authenticated, bonded, and encrypted properties individually. It is recommended to use the strongest possible security for all characteristics. It is also recommended to give the minimum level of access to characteristics needed to accomplish the required tasks. If a characteristic does not need to be writable, disable the write property. Note that, when using Bluetooth SIG services and characteristics, there will usually be requirements for each property. Applications should comply with these requirements for interoperability and to maintain qualification status. See UG438: GATT Configurator User's Guide for Bluetooth SDK v3.x for general guidance on the Bluetooth GATT Configurator.

Accessing a characteristic using a connection that does not meet the minimum requirements results in an error response being sent from the GATT server to the client. It is recommended to require encryption and authentication on all characteristics, even when the data is not sensitive.

2.1.2 Explicitly Securing Connections

Once a connection is established, two events are raised by the stack that are useful for security purposes. The first is sl_bt_evt_connection_opened. This event contains a bond handle. If the handle is set to 0xFF no bond is present. The other event of interest is sl_bt_evt_connection_parameters. This event contains the current security mode for the connection. The security modes are described in section 1.2.2 Security Modes and Levels above. The default mode is 0. Either the GATT client or server can increase the security level of the connection by calling sl_bt_sm_increase_security().

2.1.3 Security Configuration

The security configuration can be set on a device-wide basis by calling sl_bt_sm_configure(). This API is used not only to set security requirements such as MITM protection and requiring bonding for encryption but also sets the I/O capabilities for the device. This information is used when bonding, to determine which type of authentication is used during the bonding process. It is recommended to use the strongest possible security configuration and to accurately report the IO capabilities of the device. The security requirements flags are explained below. It is also recommended to require bonding for encryption, since this establishes a persistent security relationship that can prevent spoofing by devices using the same MAC address.

.

Security Feature	Description
Bonding requires MITM protection	When this flag is set, forming a new bond also requires man-in-the-middle (MITM) protection. This requires a passkey to be entered
Encryption requires bonding	Setting this flag requires a bond, rather than just pairing, for a connection to be encrypted
Require Secure Connections	Setting this flag requires that LE Secure Connections be used for exchanging cryptographic keys. This may prevent connections with older devices that do not support this feature
Bonding Requests Require Confirmation	Setting this flag requires the application to confirm any new bond request. This makes it possible to require user input for forming new bonds
Allow only Connections from Bonded Devices	Setting this flag requires that a peer device have an existing bond to successfully form a connection. This flag should be cleared to allow a new device to connect and bond.
Prefer authenticated pairing	Setting this flag causes the device to prefer authenticated pairing if both authenticated pairing and JustWorks are available.
Require SC OOB Data from both devices	Setting this flag requires secure connections out of band data to be present
Reject debug keys	Setting this this flag rejects pairing if the remote device uses debug keys.

The following table shows the different pairing or bonding methods available based on the I/O capabilities when using secure connections.

Responder	Initiator - DisplayOnly	Initiator - DisplayYesNo	Initiator - KeyboardOnly	Initiator - NoInputNoOutput	Initiator - KeyboardDisplay
DisplayOnly	Just Works	Just Works	Passkey Entry (R displays, I inputs)	Just Works	Passkey Entry (R displays, I inputs)
DisplayYesNo	Just Works	Numeric Comparison	Passkey Entry (R displays, I inputs)	Just Works	Numeric Comparison
KeyboardOnly	Passkey Entry (I displays, R inputs)	Passkey Entry (I displays, R inputs)	Passkey Entry (R and I inputs)	Just Works	Passkey Entry (I displays, R inputs)
NoInputNoOutput	Just Works	Just Works	Just Works	Just Works	Just Works
KeyboardDisplay	Passkey Entry (I displays, R inputs)	Numeric Comparison	Passkey Entry (R displays, I inputs)	Just Works	Numeric Comparison

Just Works – This method is used when one or both devices do not have a user interface that allows entering or confirming a passkey. Encryption is enabled so the connection is protected against passive eavesdropping, but MITM protection is not available since no identifying information is available. If this method is used, it is recommended to enable bonding only for short periods of time.

Passkey Entry – This method requires a passkey to be entered on one of the devices and displayed on the other. This method allows MITM to be enabled. It is recommended to allow the stack to generate random passkeys.

Numeric Comparison – This method requires that both devices display a passkey and give the user some method for confirming that the passkeys match. This is the most secure method and should be used if the I/O capabilities support it.

2.1.4 Out of Band Pairing/Bonding

Out-of-band (OOB) pairing can be used to establish a security relationship by exchanging information through a method other than Bluetooth, such as NFC. Devices that do not have a user interface can use OOB pairing to establish MITM protection through authentication and should do so whenever possible. If OOB data is not available from both devices (or legacy OOB used), the data needs to be kept secret. If OOB data is available from both devices, it does not need to be kept secret.

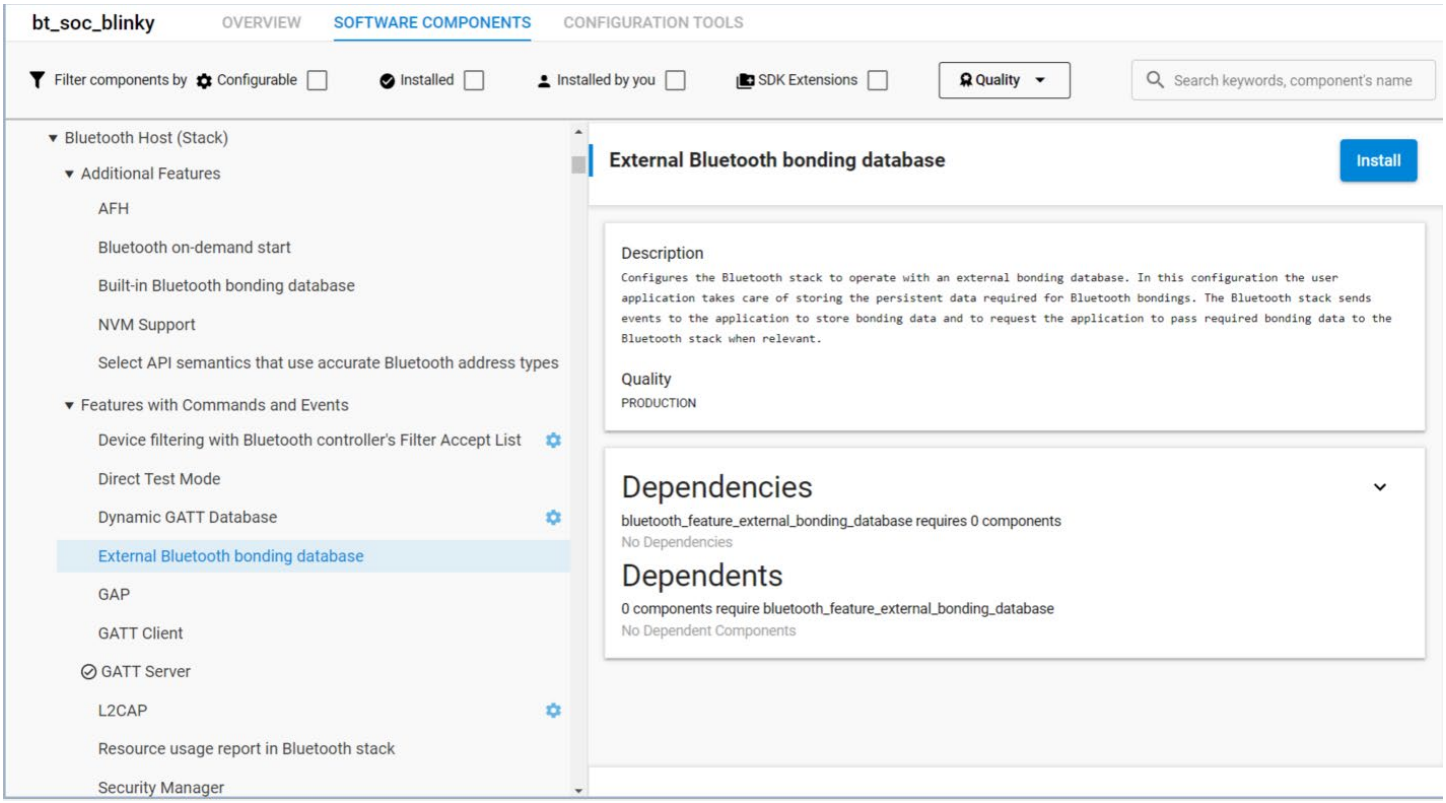
2.1.5 External bonding database

While using the external bonding database, the secure storage of the encryption keys is the responsibility of the application. For SoC case, refer to AN1271: Secure Key Storage about how to import the keys to the secure key storage.

For NCP projects the keys are usually stored on the host, so the communication between the target and the host needs to be encrypted. Silicon Labs provides two solutions for that: you can either use the “Secure NCP” feature to encrypt the serial connection, or you can use the NCP over CPC feature, and use a secured CPC channel (for details see AN1259: Using the Silicon Labs Bluetooth® Stack v3.x and Higher in Network Co-Processor Mode).

When the Bluetooth stack needs bonding data, it will send the request to the user application with a sl_bt_evt_external_bondingdb_data_request event that contains the requested data type. The application must respond to the request by sending data using the command sl_bt_external_bondingdb_set_data for setting the external bonding data.

To enable support for the external bonding database, install the External Bluetooth bonding database software component as shown here:



2.1.6 Other Stack Security Features

Disconnecting on Failed Bond Creation. The Bluetooth specification does not require a connection to be closed after a failed bonding attempt, but it is a good idea. A failed bonding may indicate that an attacker is attempting to guess the passkey. An unsuccessful bonding attempt causes the stack to raise the sl_bt_evt_bonding_failed event. This event indicates the connection used in the failed attempt as well as a reason for the failure. The status codes are documented in the Gecko Platform API Reference. The connection can be closed with a call to sl_bt_connection_close().

Disable Bonding Except When Needed. Leaving bondable mode enabled always is unnecessary and may make it easier for an attacker to establish a trusted relationship with your device. For this reason, it is recommended to limit the length of time that new bonds can be

formed, either a short duration after power-up or by requiring some user action such as a button press to enable bondable mode. The bondable mode is set by a call to sl_bt_set_bondable_mode().

Minimum Key Size for Bonding. It is possible to set the minimum key size required for bonding using sl_bt_sm_set_minimum_key_size(). It is recommended to use the default, largest setting of 16 bytes.

Bonding Configuration. The maximum number of bonds can be set from 1 to 32. The bonding policy determines what happens when the bonding table is full. It is recommended to use policy 0 – new bonding attempts fail. It is better for the application to explicitly delete bonds that are no longer needed.

2.2 Coexistence with Wi-Fi

Designs using PTA for managed coexistence with Wi-Fi should take precautions against attacks against the PTA interface. Such attacks may include Denial-of-service and information disclosure. If the Wi-Fi PTA master is compromised, channel access may never be granted. One possible to solution is to disable the PTA interface if channel access is denied for extended periods of time. The coexistence options are documented in the coexistence section of the Bluetooth API Reference. The coexistence API includes counters to determine the number of times channel access is denied or aborted and is documented in this section of the Bluetooth API Reference. See AN1128: Bluetooth Coexistence with Wi-Fi for more detailed information.

3 Privacy and Tracking

This section discusses privacy for mobile Bluetooth Low Energy devices.

Many Bluetooth Low Energy devices will be wearable or other mobile devices. If the device is constantly advertising the same address, it is easy for the device's location to be tracked.

3.1 Bluetooth Address Types

The Bluetooth specification defines several address types that are defined below. All Bluetooth addresses are 48 bits in length.

Identity Address – a type of address that can be used in forming a new bond. Either public addresses or random static addresses qualify as identity addresses.

Public address – A device's public address is usually derived from the hardware and does not change over time.

Resolvable private address – This address type is random but can be resolved by a bonded device in possession of an identity-resolving key (IRK).

Non-resolvable private address – This is a completely random address that is only used in non-connectable advertising. The identity address cannot be determined by observers if they only know the non-resolvable private address.

Random Static address – All but the two upper-most bits are random. A new random static address may be chosen after a power cycle event.

Anonymous – Bluetooth 5.x style extended advertising supports the ability to advertise without any address being sent along with the advertising payload.

3.1.1 LE Privacy

LE Privacy is a feature that was introduced in Bluetooth 4.1. When this feature is enabled, a new resolvable private address is chosen periodically by the stack. LE Privacy can be enabled by calling sl_bt_gap_set_privacy_mode(). It is recommended to use this feature on devices where tracking is a concern.

3.1.2 Device Privacy

When a device is in device privacy mode, it is only concerned about its own privacy. It should accept advertising packets from peer devices that contain their Identity Addresses as well as their private address, even if the peer device has distributed its IRK.

3.1.3 Network Privacy

When a device is in network privacy mode, it shall not accept advertising packets containing the Identity Address of peer devices that have distributed their IRK; that is, only resolvable private address (RPA) is accepted for peer devices that have distributed their IRK.

If the Resolvable Private Address Only characteristic is not present in the GAP service of the remote device, it may use its Identity Address over the air.

3.1.4 Working with LE Privacy

3.1.4.1 Address Resolving List

Silicon Labs Bluetooth stack maintains an address resolving list for devices using LE privacy. Devices can be referred to by the bonding handle or by address. The API reference is found at https://docs.silabs.com/bluetooth/6.1.0/bluetooth-stack-api/sl-bt-resolving-list.

- Bonding Handle

Adding a device to the address resolving list requires the bonding handle and privacy mode of the device. The privacy mode setting indicates whether the local device uses device privacy or network privacy for the remote device. Removing a device through its bonding handle only requires the bonding handle. Note: deleting the bonding does not remove the device from the address resolving list nor from the filter accept list.

- Address

Adding a device to the address resolving list by its address requires the identity address of the device, the type of address, device's IRK, and the privacy mode. The privacy mode setting indicates whether to use device privacy or network privacy for the peer device. Removing a device from the address resolving list requires the address to be removed and the address type.

3.1.4.2 Filter Accept List and Advertisement Filtering

Silicon Labs' Bluetooth stack maintains a filter accept list which can used to filter out advertisements from any device that is not in the list. As with the address resolving list, devices can be added and removed from the filter accept list either by bonding handle or by address. The filtering policy can be set by using the sl_bt_scanner_set_parameters_and_filter() API function which is documented here https://docs.silabs.com/bluetooth/6.1.0/bluetooth-stack-api/sl-bt-scanner#sl-bt-scanner-set-parameters-and-filter.

- Bonding Handle

Devices can be added to the filter accept list using the bonding handle.

- Address

Devices can be added to the filter accept list using the peer address and the address type.

4 Application Layer Security

Relying entirely on Bluetooth security features also means relying on the security of the peer device's Bluetooth stack. Doing so introduces additional unknowns. As well as the potential of unpatched vulnerabilities in an unknown stack, the remote device may share the Bluetooth connection with multiple applications. This can lead to compromised integrity and/or confidentiality. The solution to this problem is for the application to form a secure link on top of the Bluetooth connection by establishing an authenticated, shared secret key and encrypting data with that key before sending it over the Bluetooth connection. This section discusses the key components involved in using application layer security: authentication, key agreement, key derivation, and encryption.

4.1 Secure Identity

A common problem with small IoT devices is that they have no user interface to authenticate the device's identity, such as entering or confirming a passkey. A device that cannot be authenticated is unable to take advantage of man-in-the-middle (aka machine-in-the-middle) protection. One solution to this problem is to use secure identity certificates to uniquely identify a device. A signed certificate chain, along with the corresponding private signing key, is stored on the device. The device's certificate chain can be sent over the Bluetooth LE GATT to the peer device to authenticate its identity. More information on secure identity certificates is available in AN1268: Authenticating Silicon Labs Devices using Device Certificates. The communication flow is illustrated in the following figure.

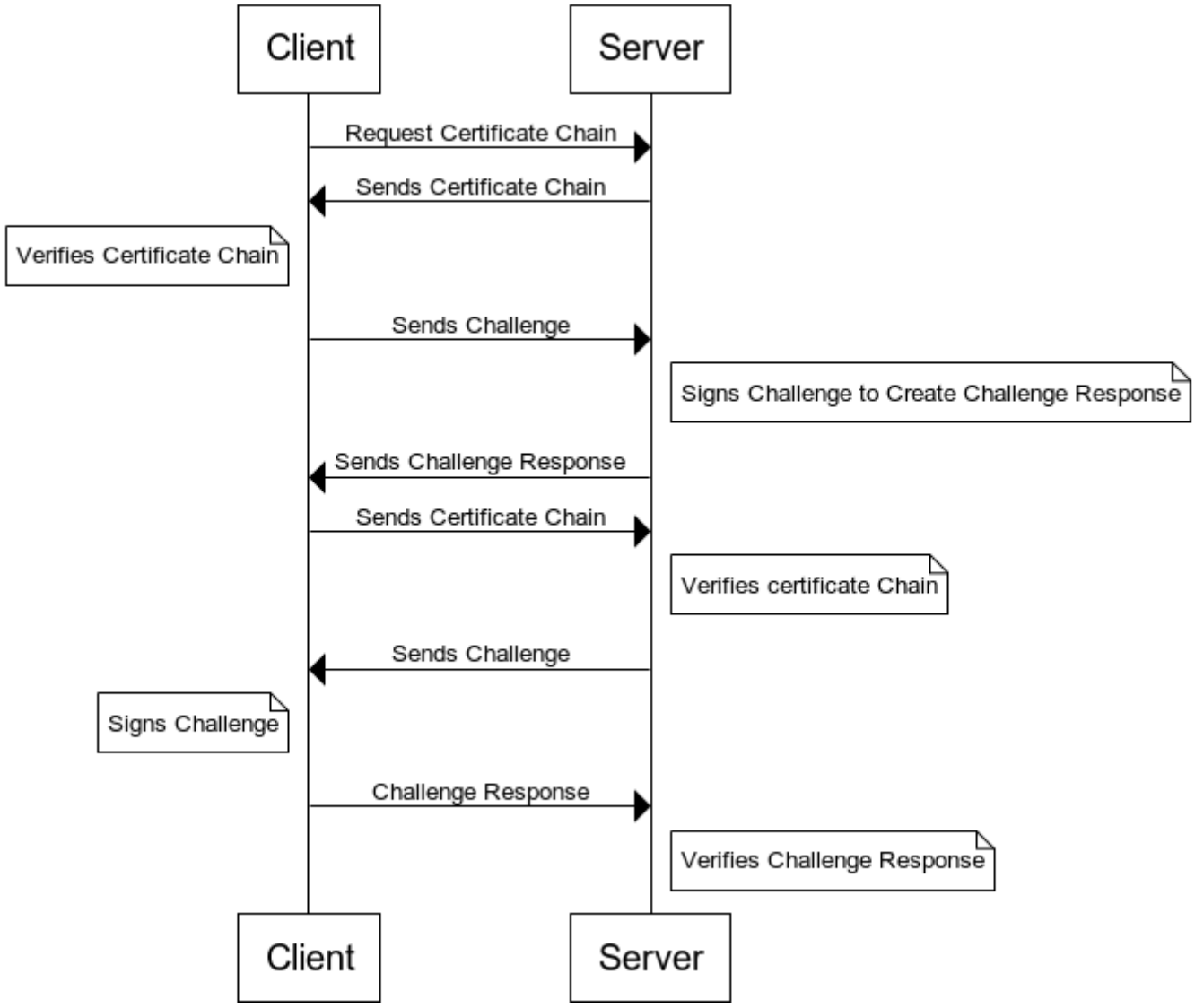


Figure 4-1. Exchanging Certificate Chains to Authenticate Device Identity

4.1.1 Key Agreement

An important step in securing data at the application is for the two peer devices is to establish a shared secret. The accepted method for doing so is by ECDH as mentioned earlier. A brief description is found at https://en.wikipedia.org/wiki/Elliptic-curve_Diffie%E2%80%93Hellman. To perform ECDH over Bluetooth Low Energy, GATT characteristics can be set up to store the public key for each device and then the keys exchanged like any other data. To ensure that keys have come from a trusted source, they can be signed with the sender's secure identity key. Once the public keys have been exchanged, a shared secret can be computed. A key derivation function is usually applied rather than using the shared secret directly.

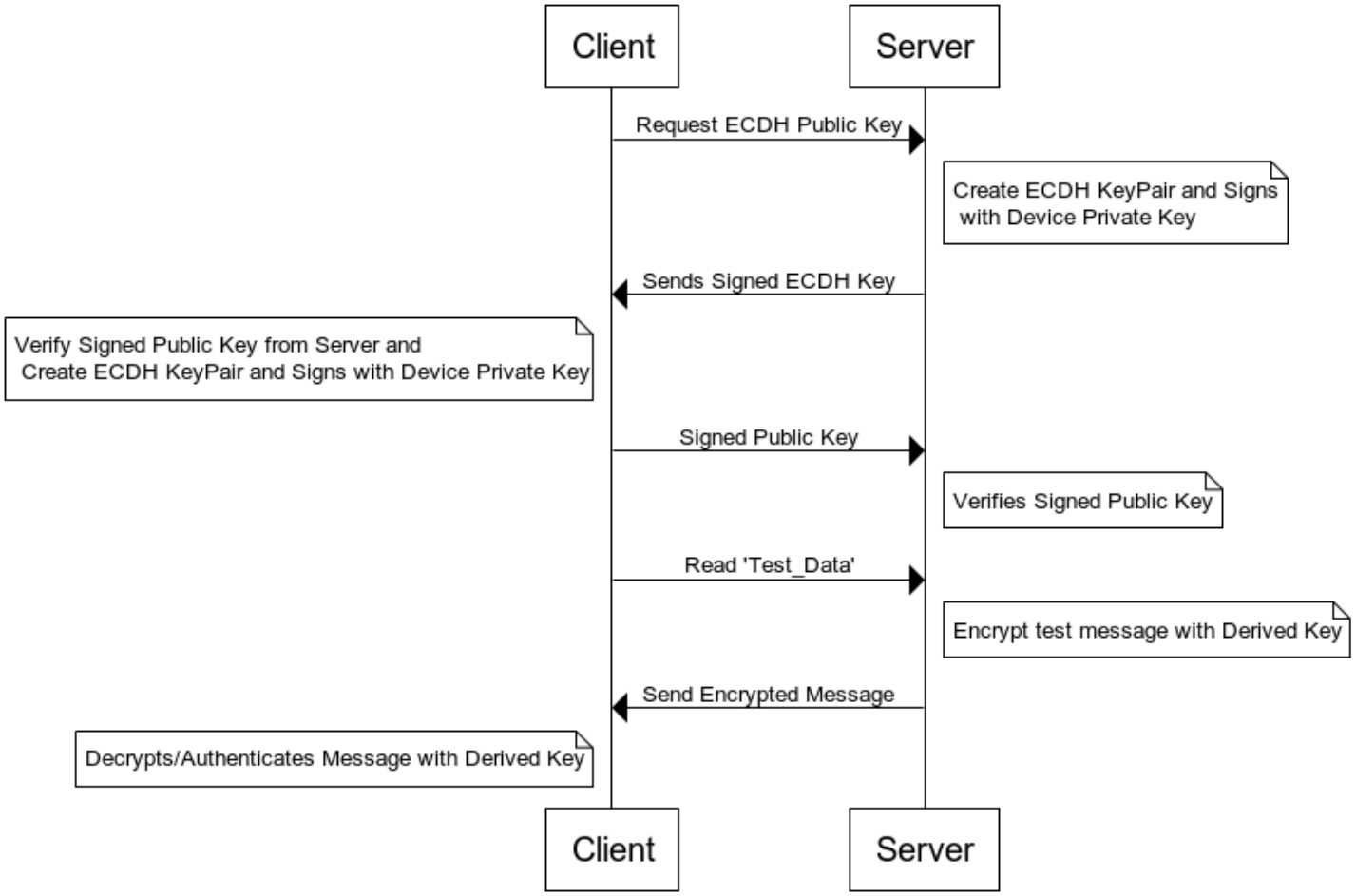


Figure 4-2. Authenticated Key Agreement

4.1.2 Key Derivation Function

A key derivation function takes a shared secret and transforms it to introduce some additional entropy, or randomness. A typical method for deriving keys is by using a cryptographically-secure hash such as SHA-2/256. Silicon Labs series 2 devices also provide hardware acceleration for other key derivation functions such as PBKDF2 and HKDF. Derived keys with persistent storage should be stored using the most secure method available. Secure Vault High parts can wrap, or encrypt, keys. See AN1271: Secure Key Storage for more information on this topic.

4.2 Application Layer Encryption

This section describes the encryption of user data before it is passed to the Bluetooth stack to be sent to a peer device.

4.2.1 Encrypting and Authenticating Data

Once a shared secret has been established and symmetric keys derived, sensitive data can be encrypted/decrypted and authenticated. Several cipher modes are accelerated in hardware.

Counter with Cipher Block Chaining Mode (CCM). This cipher mode combines a counter with CBC-MAC to provide several benefits. It is possible to authenticate both encrypted and unencrypted data in the same message, known as authenticated encryption of associated data (AEAD). A unique initialization vector is required for each message sent. A brief description is found at https://en.wikipedia.org/wiki/CCM_mode. When this mode is used, the data, initialization vector and authentication tag can be sent together in a single block

Galois Counter Mode (GCM). This cipher mode is like CCM described above but uses a different method for authenticating data and lends itself to parallelization. A brief description is found at https://en.wikipedia.org/wiki/Galois/Counter_Mode. As with CCM, the initialization vector, data and authentication tag can be sent together in a single block.

4.2.2 Encryption-Only Cipher Modes

Encryption-only cipher modes are also supported in Silicon Labs series 2 devices in case authentication is not desired. The reference manual for your chosen device documents all cipher modes accelerated in hardware.

4.2.3 Application Layer Encryption Example

The example accompanying this application note, available on https://github.com/SiliconLabs/bluetooth_applications, demonstrates a method for implementing application layer encryption with authenticated key exchange using device identity certifications. Two projects are included: one for the client and the other for the server. The communication flow is summarized in the following table.

Client	Initiator	Server
Certificate Chain Exchange		
Scans for devices advertising Secure Attestation service	→ ←	Advertises Secure Attestation service
Connects to server and requests connection be encrypted with JustWorks pairing	→ ←	Accepts and encrypts connection
Reads Device and Batch certificates from Server	→ ←	Sends Device and Batch certificates
Verifies server's certificate chain and sends random challenge	→ ←	Signs random challenge with private device key and returns signature
Verifies the server's signature, saves the public key, and sends Device and Batch certificates	→ ←	Saves Device and Batch certificate, verifies client's certificate chain, and sends random challenge
Signs random challenge and returns signature	→	Verifies client's signature, and saves the public key.
ECDH Key Exchange		
Generates ECDH keypair and signs with private device key		Generates ECDH keypair and signs with private device key
Requests signed ECDH public key from server	→ ←	Sends signed ECDH public key to client
Receives and verifies Server's signed public key. Sends signed public key to server	→ ←	Receives and verifies Client's signed public key
AES Operation		
Derives AES key using SHA2/256-based function		Derives AES key using SHA2/256-based function
Requests encrypted message from server	→ ←	Encrypts sample message using AES key and sends to client
Decrypts message using AES key		Idle

The characteristics used in this communication flow are summarized the following table.

Table 4-1. Secure Attestation Characteristics

Attribute Name	Description
Device Certificate	The server's device identity certificate
Batch Certificate	The server's batch certificate
Peer Device Certificate	The client's device identity certificate
Peer Batch Certificate	The client's batch certificate
Server public key	Server's ECDH public key
Client public key	Client's ECDH public key
Challenge	Random challenge
Response	Signature of the random challenge
Test_data	Test data to encrypt

4.3 Other Application Security Concerns

Beyond using the security features incorporated into the Bluetooth specification itself, an application can take several steps to implement higher security.

A common type of attack relies on buffer overflows. For this reason, it is highly recommended that applications validate the size of data being received before attempting to store it. Whether a block of data is received through a notification/indication, write or a response to a

read request, the size of the data received is always reported by the stack. The size parameter must be used to determine if the application has reserved sufficient space to store it.

4.4 Secure Boot

The ability to ensure that authentic firmware is running on the device is possibly one of the most valuable security features available. Many attacks rely on the attacker's ability to gain control of the device by some method such as remote code injection. Silicon Labs' secure boot with root of trust secure loader (RTSL) provides additional security by verifying the authenticity of the firmware using an ECDSA digital signature and an immutable public key stored on the device. See AN1218: Series 2 Secure Boot with RTSL for specifics on using secure boot.

4.5 Signed and Encrypted Firmware Updates

One concern with over-the-air (OTA) updates is that the firmware could be stolen by sniffing the Bluetooth connection, since the Silicon Labs AppLoader does not support encrypted connections. The solution is to encrypt the firmware image itself and allow the bootloader on the device to be updated to perform the decryption. Firmware images can also be signed to prevent inauthentic images from being flashed into active memory in the first place. The signature of the update images is verified using the device public signing key.

4.6 Subscribe to Security Advisories

Silicon Labs recommends that all customers subscribe to security advisories to be aware of the latest threats and mitigations. See www.silabs.com/security for instructions.

4.7 Additional Reading

AN1190: Secure Debug

AN1218: Secure Boot with RTSL

AN1222: Production Programming of Series 2 Devices

AN1247: Anti-Tamper Protection Configuration and Use

AN1268: Authenticating Silicon Labs Devices Using Device Certificates

AN1271: Secure Key Storage

UG103.05: IoT Endpoint Security Fundamentals

Simplicity Studio

One-click access to MCU and wireless tools, documentation, software, source code libraries & more. Available for Windows, Mac and Linux!

IoT Portfolio
www.silabs.com/IoT

SW/HW
www.silabs.com/simplicity

Quality
www.silabs.com/quality

Support & Community
www.silabs.com/community

Disclaimer
Silicon Labs intends to provide customers with the latest, accurate, and in-depth documentation of all peripherals and modules available for system and software implementers using or intending to use the Silicon Labs products. Characterization data, available modules and peripherals, memory sizes and memory addresses refer to each specific device, and "Typical" parameters provided can and do vary in different applications. Application examples described herein are for illustrative purposes only. Silicon Labs reserves the right to make changes without further notice to the product information, specifications, and descriptions herein, and does not give warranties as to the accuracy or completeness of the included information. Without prior notification, Silicon Labs may update product firmware during the manufacturing process for security or reliability reasons. Such changes will not alter the specifications or the performance of the product. Silicon Labs shall have no liability for the consequences of use of the information supplied in this document. This document does not imply or expressly grant any license to design or fabricate any integrated circuits. The products are not designed or authorized to be used within any FDA Class III devices, applications for which FDA premarket approval is required or Life Support Systems without the specific written consent of Silicon Labs. A "Life Support System" is any product or system intended to support or sustain life and/or health, which, if it fails, can be reasonably expected to result in significant personal injury or death. Silicon Labs products are not designed or authorized for military applications. Silicon Labs products shall under no circumstances be used in weapons of mass destruction including (but not limited to) nuclear, biological or chemical weapons, or missiles capable of delivering such weapons. Silicon Labs disclaims all express and implied warranties and shall not be responsible or liable for any injuries or damages related to use of a Silicon Labs product in such unauthorized applications.
Note: This content may contain offensive terminology that is now obsolete. Silicon Labs is replacing these terms with inclusive language wherever possible. For more information, visit www.silabs.com/about-us/inclusive-lexicon-project

Trademark Information
Silicon Laboratories Inc.®, Silicon Laboratories®, Silicon Labs®, SiLabs® and the Silicon Labs logo®, Bluegiga®, Bluegiga Logo®, EFM®, EFM32®, EFR, Ember®, Energy Micro, Energy Micro logo and combinations thereof, "the world's most energy friendly microcontrollers", Redpine Signals®, WiSeConnect , n-Link, ThreadArch®, EZLink®, EZRadio®, EZRadioPRO®, Gecko®, Gecko OS, Gecko OS Studio, Precision32®, Simplicity Studio®, Telegesis, the Telegesis Logo®, USBXpress® , Zentri, the Zentri logo and Zentri DMS, Z-Wave®, and others are trademarks or registered trademarks of Silicon Labs. ARM, CORTEX, Cortex-M3 and THUMB are trademarks or registered trademarks of ARM Holdings. Keil is a registered trademark of ARM Limited. Wi-Fi is a registered trademark of the Wi-Fi Alliance. All other products or brand names mentioned herein are trademarks of their respective holders.

SILICON LABS

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

www.silabs.com