Find My Network Accessory Specification

Developer Preview

Release R1

Developer

Contents

1. Introduction

This document is a Developer Preview, to be used for informational purposes only. Developers intending to develop or manufacture a Find My network-enabled accessory must be enrolled in the MFi Program. Although its content has been reviewed for accuracy, it is not final and is subject to change. Apple is supplying this content to help accessory developers plan for the adoption of the feature(s) described herein.

1.1. Requirements, recommendations, and permissions

This specification contains statements that are incorporated by reference into legal agreements between Apple and its Licensees. The use of the words *must, must not, required, shall, shall not, should, should not, recommended, not recommended, may, optional,* and *deprecated* in a statement have the following meanings:

- *Must, shall*, or *required* means the statement is an absolute requirement.
- *Must not, shall not,* or *prohibited* means the statement is an absolute prohibition.
- *Should* or *recommended* means the full implications must be understood before choosing a different course.
- *Should not* or *not recommended* means the full implications must be understood before choosing this course.
- *May* or *optional* means the statement is truly optional, and its presence or absence cannot be assumed.
- *Deprecated* means the statement is provided for historical purposes only and is equivalent to "must not."

The absence of requirements, recommendations, or permissions for a specific accessory design in this specification must not be interpreted as implied approval of that design. Licensee is strongly encouraged to ask Apple for feedback on accessory designs that are not explicitly mentioned in this specification.

1.2. Terminology

Throughout this document, these terms have specific meanings:

- The term *Apple device* is used to refer to an iPhone, iPad, iPod, Apple Watch, or Mac (running iOS, iPadOS, watchOS, or macOS).
- The term *accessory* is used to refer to any product intended to interface with a device through the means described in this specification.
- The term *Apple ID* is an authentication method that Apple uses for iPhone, iPad, Mac, and other Apple devices and services. When an Apple ID is used to log in to an Apple device, the device will automatically use the settings associated with the Apple ID.

2. Core Concepts

2.1. Overview

The *Find My Network Accessory Specification* defines how an accessory communicates with Apple devices to help owners locate their accessories privately and securely by using the Find My network.

2.2. Find My app

The Find My app is where you locate your Apple devices, share your location with friends and family, and locate all Find My network-enabled accessories. The app displays the location of findable items and includes additional features to protect your devices, such as playing sound, using Lost Mode, and so on. See the Find My webpage for more details.

2.3. Transport

The Find My network accessory protocol uses Bluetooth Low Energy (BTLE) as the primary transport to interact with Apple devices.

2.4. Operation

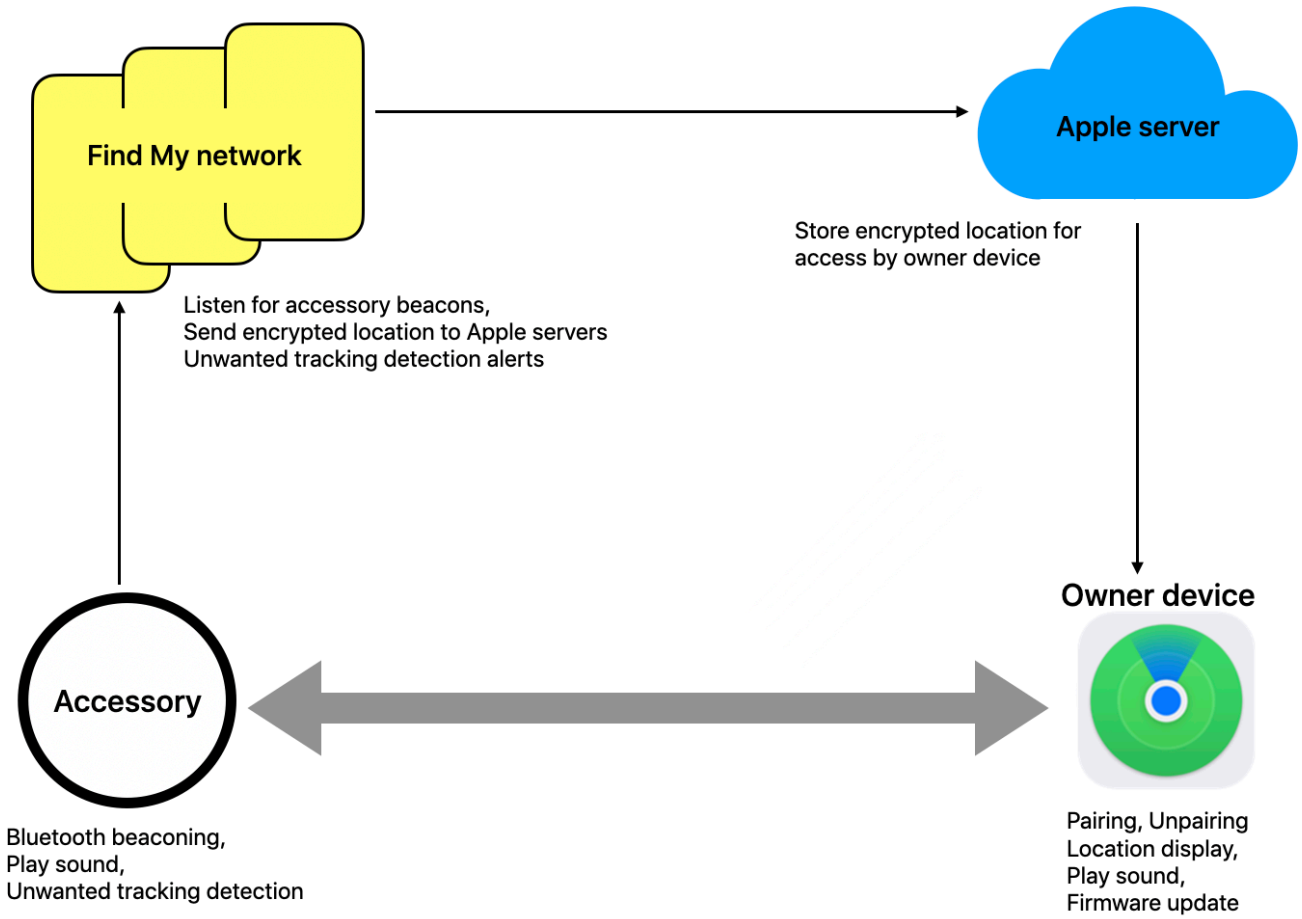
The accessory and the owner Apple device generate a cryptographic key pair after Find My network pairing. The owner Apple device has access to both the private and the public key, and the accessory has the public key.

An accessory subsequently broadcasts a rotating key (derived from the public key) as a low energy Bluetooth beacon. This beacon can be picked up by nearby Apple devices (see Find My network). The Apple devices publish the key received in the Bluetooth beacon, along with its own location encrypted using that same key, to Apple servers. Because the private key is stored only on the owner device, the location information is accessible only to the device owner. The data stored in Apple servers is end-to-end encrypted, and Apple does not have access to any location information.

2.5. Roles

2.5.1. Owner device

When an accessory is paired with an Apple device through the Find My app, the accessory is associated with the Apple ID on that device. This device and all other Apple devices signed in with the same Apple ID are treated as owner devices.



The Find My app on an owner device can be used to locate accessories. An owner device is required for actions such as unpairing the device, firmware update, locate, and so on.

2.5.2. Accessory

An *accessory* is the device that implements the Find My network accessory protocol and can be located using the Apple Find My network and servers. The accessory is paired with the Apple ID in use on the owner device.

2.5.3. Find My network

The Find My network provides a mechanism to locate accessories by using the vast network of Apple devices that have Find My enabled. When an accessory is detected by a nearby Apple device, the device publishes its own encrypted location as the approximate location of the detected accessory. Reports from more than one Apple device can provide a more precise location. Any Apple device that participates in the Find My network is called a *Finder device*. Participation in the Find My network is a user choice that can be reviewed or changed anytime in Settings.

A *non-owner device* is an Apple device in a Find My network that may connect to the accessory but is not an owner device. (For example, a device might connect in response to a UT alert; see Unwanted tracking detection.)

2.5.4.Apple server

Apple server receives encrypted location data from Finder devices and temporarily stores it. Only the owner devices can decrypt and read raw locations from the encrypted data. Apple cannot read this information.

2.6. Features

2.6.1. Unwanted tracking detection

Unwanted tracking detection (UT) notifies the user of the presence of an unrecognized accessory that may be traveling with them over time and allows them to take various actions, including playing a sound on the accessory if it's in BTLE range.

2.6.2.Lost mode

An owner can use the Find My app to place their accessory in Lost Mode. They can set a phone number and select a message from a predefined list.

When someone finds someone else's lost accessory, they can discover the details set by the owner by using NFC or BTLE to help the owner recover the lost item. Details on the BTLE protocol and its security requirements will be provided in an updated developer preview.

2.6.3.Play sound

The Play sound feature allows users to play sound from their Apple device to locate the accessory. This action may be initiated from an owner or non-owner device.

Users can play a sound from the Find My app on an owner device. The Apple device creates a BTLE connection or uses an existing connection to the accessory and uses the Play sound—owner control point to initiate the action.

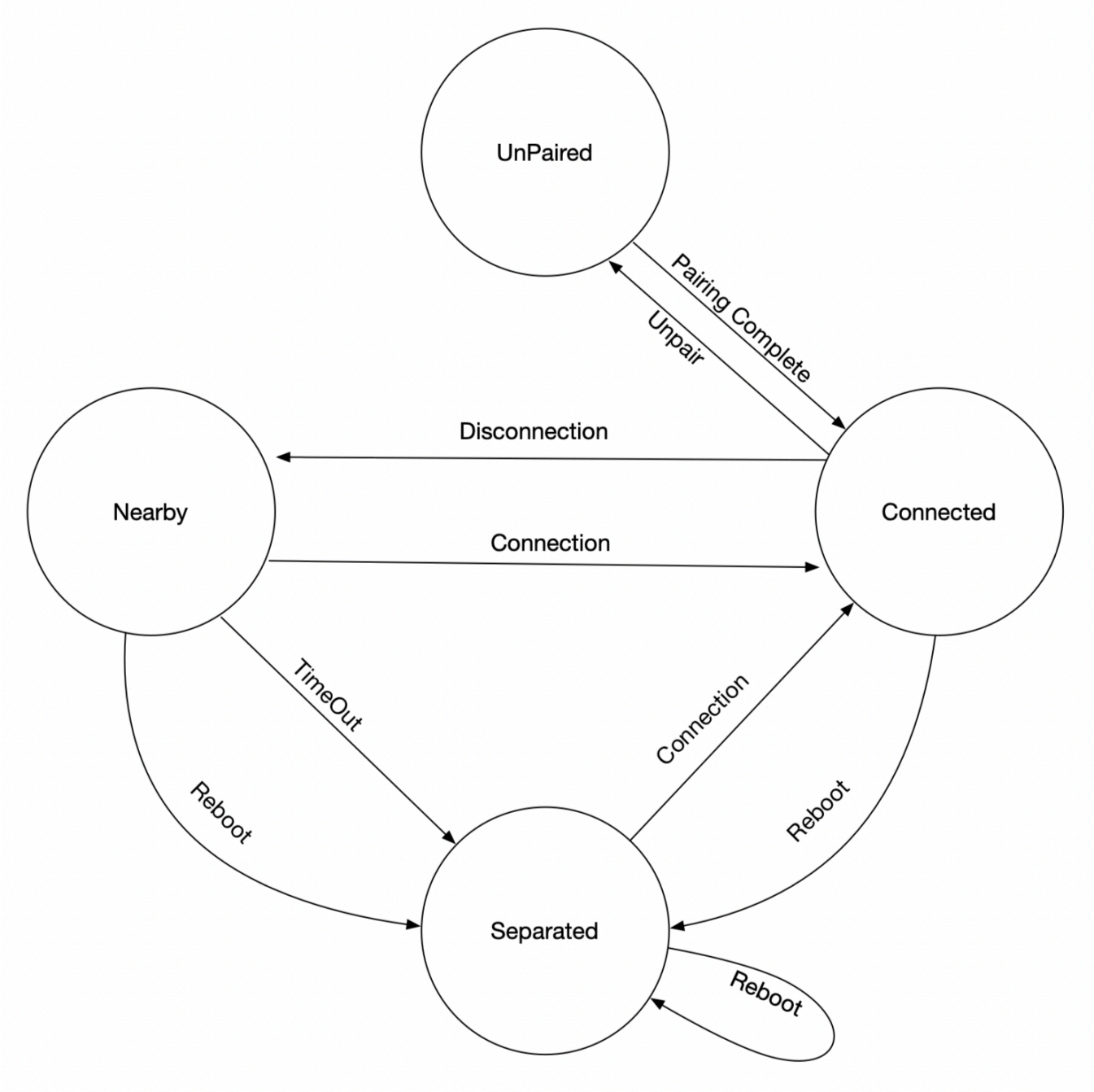
Users can play a sound from a non-owner Apple device when a UT alert appears on that device. The Apple device creates a BTLE connection and uses the Play sound—non-owner control point to initiate the action.

2.7. States

Accessory operations can be described using a state machine with the states listed in this section and transition between them based on interactions with an owner device.

2.7.1. Unpaired

The accessory must be in an unpaired state on first startup or before the accessory setup is completed. In this state, the accessory must advertise Find My network service as a primary service in a



connectable Bluetooth advertisement (See BTLE advertising). The owner user initiates pairing from an owner device. See Pairing for the pairing procedure.

2.7.2. Connected

The accessory must enter connected state after the Find My network pairing successfully completes with the owner device. The owner device may disconnect from the accessory after pairing completes. Once paired, the accessory must not pair with another Apple device for Find My network functions. It must stay paired until it successfully completes the unpairing procedure with the owner device.

The accessory must reenter the connected state from nearby or separated state or whenever an owner device connects to the accessory. The accessory shall support simultaneous connections to two Apple devices on the same iCloud account.

Motion detection and UT protocols are disabled in connected state. When the accessory enters this state, advertising payload is set to the nearby key.

2.7.3. Nearby

The accessory must enter the nearby state immediately after it disconnects from an owner device. The accessory shall remain in nearby state for T_{NEARBY}. See Timers and constants.

Motion detection and unwanted tracking detection protocols are disabled in nearby state. When the accessory enters this state, advertising payload is set to the nearby key. See Payload for nearby state for details.

2.7.4. Separated

The accessory must enter the separated state under these conditions:

1. The accessory is paired and starts up from a reset, power cycle, or other reinitialization procedure.
2. The accessory is in nearby state and the T_{NEARBY} time-out has expired.

Motion detection and unwanted tracking detection protocols are enabled in separated state. When the accessory enters this state, advertising payload is set to the separated key. See Payload for separated state for details.

3. Requirements

3.1. Overview

Accessories that support the Find My network accessory protocol must conform to the requirements listed in this chapter, along with any feature-specific requirements contained in other chapters.

3.2. General

An accessory that supports the Find My network accessory protocol must meet these requirements:

- It must incorporate software authentication. See Software authentication for details.
- It must enable the user to set up the accessory using the Apple Find My app, both out of the box and after every factory reset, without requiring additional setup procedures.
- It must be certified and listed as a qualified end product by the Bluetooth SIG.
- It must not operate simultaneously on the Find My network and other finder network, or otherwise implement functionality which may interfere with the security and privacy requirements referenced in this document.

3.3. Hardware

3.3.1. Bluetooth

The accessory must use a Bluetooth controller that meets the following requirements:

- LE 2M uncoded PHY
- Random resolvable addresses
- Data packet length extension
- LE advertising extensions

For details, refer to the *Bluetooth Core Specification Version 5.2 Feature Overview*.

3.3.2.Product-specific requirements

During separated UT state, motion-triggered UT sound alerts from the accessory is designed to bring awareness to the person with whom it's detected. These alerts are created using a sound maker (for example, a speaker) and a motion detector (such as an accelerometer).

Products whose primary purpose is to help find items, or are a size that would make it difficult to be discovered by the person in possession of it, must include a sound maker and a motion detector to support motion-triggered UT sound alerts. See Unwanted tracking detection for detailed requirements.

Additional considerations and requirements for size, and for products whose purpose is not to help find items, will be provided in an updated developer preview.

3.3.3.Serial number lookup

The serial number of the accessory must be printed on it. The number must be unique for each product ID and must be readable either through NFC Tap (see additional requirements under NFC) or BTLE. For privacy reasons, a serial number must be readable over BTLE only when a device is unpaired or upon user action on the accessory (for example, when pressing and holding a button).

Details on the BTLE protocol and its security requirements will be provided in an updated developer preview.

3.3.4.Disable

An accessory must have a mechanism to disable Find My network (for example, a power off button, or battery removal) based on user intent.

Additional requirements will be provided in an updated developer preview.

3.3.5.Clock accuracy

The accessory must support 15-minute key rotation using hardware timers. Apple devices expect the accessory to have a 200 PPM oscillator, causing a potential drift rate of 17.28 seconds per day. See Timers and constants for more details. On every connection, the owner device sends the Configure separated state command. The accessory shall synchronize its clock using the NextPrimaryKeyRoll parameter of the `Configure_Separated_Mode` command.

3.4. Cryptography

3.4.1. Operations

Pairing the accessory with an owner device as well as deriving keys requires the following:

- A cryptographically secure DRBG (see NIST Special Publication 800-90A) with a reliable source of entropy (see NIST Special Publication 800-90B).
- Modular reduction and addition of big integers.
- An implementation of the SHA-256 cryptographic hash function.
- An implementation of the ANSI x9.63 KDF (see SEC1, 3.6.1 ANSI X9.63 Key Derivation Function).
- Computations on the NIST P-224 elliptic curve (see FIPS 186-4, D.1.2.2. Curve P-224):
  - Generation of a random scalar in [1, q).

  - Scalar multiplication and point addition.
  - Verification that a point is on the P-224 elliptic curve.
- ECDSA/ECDH over the NIST P-256 elliptic curve (see FIPS 186-4, D.1.2.3. Curve P-256 and Pairing for more details).
- AES-128-GCM decryption (see NIST Special Publication 800-38D).

3.4.2. Implementation

Cryptographic operations and algorithms must compute on secret values in constant time to defend against timing attacks. Similarly, a secret value (or parts of one) must not be used as a memory offset or as the condition for a branch instruction.

Scalar generation should either use rejection sampling or generate at least 64 more bits than needed so that the bias due to the modular reduction is negligible (see FIPS 186-4, B.4.1 Key Pair Generation Using Extra Random Bits and B.4.2 Key Pair Generation by Testing Candidates). The scalar must not be generated by simply reducing the minimally required number of random bytes modulo q (the order of the base point) because this leads to a biased distribution.

Implementation of the scalar multiplication and point addition on elliptic curves must be safe against timing attacks. An exception may be made when computing on public values; for example, to speed up ECDSA signature verification. A variable-time, double-base scalar multiplication for ECDSA signature verification must not be used to compute primary or secondary keys.

Upon receiving a scalar, it must be checked to be in range `[1, q)`, where q is the order of the base point of the elliptic curve, before continuing with the protocol flow. See Scalar validation.

Upon receiving an elliptic curve point, it must be checked to be on the curve. See Elliptic curve point validation.

3.4.2.1. Endianness and wire format

All elliptic curve points, coordinates, and scalars must be transmitted in big-endian byte order; that is, the most significant bytes are sent first.

Whenever a scalar or a coordinate is the input for an algorithm like SHA-256() or `ANSI-X9.63-KDF()`, or the output of a function, its byte order is assumed to be big-endian. A point is expected to be formatted in uncompressed ANSI X9.63 format. See SEC1, 2.3.3 EllipticCurvePoint-to-OctetString Conversion.

3.4.2.2. Random scalar generation

Whenever this specification requires generation of a P-224 scalar, follow this process:

1. Generate r = 28 random bytes using a cryptographically secure DRBG. See Operations.
2. If r >= q - 1, where q is the order of the base point of the P-224 elliptic curve, go to step 1.
3. Compute s = r + 1 and return s as the new scalar.

Another option to securely generate a P-224 scalar is as follows:

1. Generate r = 36 random bytes using a cryptographically secure DRBG. See Operations.
2. Compute k = r (mod q-1), where q is the order of the base point of the P-224 elliptic curve.

3. Compute s = k + 1 and return s as the new scalar.

Whenever this specification requires generation of a P-256 scalar, follow this process:

1. Generate r = 32 random bytes using a cryptographically secure DRBG. See Operations.
2. If r >= q - 1, where q is the order of the base point of the P-256 elliptic curve, go to step 1.
3. Compute s = r + 1 and return s as the new scalar.

Another option to securely generate a P-256 scalar is as follows:

1. Generate r = 40 random bytes using a cryptographically secure DRBG. See Operations.
2. Compute k = r (mod q-1), where q is the order of the base point of the P-256 elliptic curve.
3. Compute s = k + 1 and return s as the new scalar.

3.4.2.3. Scalar validation

Whenever this specification requires validation of a P-224 scalar, follow this process:

1. If the given scalar s = 0, reject it as invalid.
2. If s >= q, where q is the order of the base point of the P-224 elliptic curve, reject s as invalid.
3. Make s a valid scalar.

3.4.2.4. Elliptic curve point validation

Whenever this specification requires validation of a P-224 elliptic curve point, follow this process:

1. Check that the length of a point is 57 bytes.
2. Decode x and y as big-endian integers in the range $[0, 2^{224})$.
3. Check that $x < p$ and $y < p$, where $p = 2^{224} - 2^{96} + 1$.
4. Check that $y^2 = x^3 + ax + b$, where $a = p - 3$ and $b =$ 0xb4050a850c04b3abf54132565044b0b7d7bfd8ba270b39432355ffb4.

3.4.2.5. ECDSA signature verification

Whenever this specification requires verification of a P-256 ECDSA signature over a message m:

1. Decode the given signature in X9.62 format to obtain two 32-byte big-endian integers r and s (see SEC1, C.5 Syntax for Signature and Key Establishment Schemes).
2. Check that $0 < r < p$ and $0 < s < p$, where $p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$.
3. Compute $e = SHA{-}256(m)$, where m is the signed message.
4. Let z be the |q| leftmost bits of e, where |q| is the bit length of the group order q.
5. Compute $u_1 = zs^{-1} \pmod q$ and $u_2 = rs^{-1} \pmod q$.
6. Compute the point $(x, y) = u_1 \cdot G + u_2 \cdot Q_A$, where G is the base point of the P-256 elliptic curve and Q_A is Apple's signature verification key.
7. If (x, y) is the point at infinity, reject the signature.
8. If $r = x \pmod q$, then accept the signature, and if not, reject it.

See Apple server public keys for signature verification key (Q_A) details.

3.4.2.6. ECIES encryption

Whenever this specification requires encryption of a message M to a P-256 public key $P=Q_E$ (Apple server encryption key), follow this process:

1. Generate an ephemeral P-256 scalar k as described in Random scalar generation.
2. Compute the public point Q = k · G, where G is the base point of the P-256 elliptic curve.
3. Compute the shared secret Z = k · P.
4. Derive 32 bytes of keying material as V = ANSI–X9.63–KDF(x(Z), Q || P).
5. Set K = V[0..15], that is, the first 16 bytes of the keying material V.
6. Set IV = V[16..31], that is, the last 16 bytes of the keying material V.
7. Encrypt message M as (C,T) = AES-128-GCM(K, IV, M) without any additional authenticated data. K is the 128-bit AES key, IV is the initialization vector, C is the ciphertext, and T is the 16-byte authentication tag.
8. Output Q || C || T; that is, the ephemeral public key Q concatenated with the ciphertext and the authentication tag.

See Apple server public keys for the Apple server's encryption key (Q_E) details.

3.4.2.7. AES-GCM decryption

Whenever this specification requires AES-128-GCM decryption of a message M, given a 128-bit AES key K, follow this process:

1. Decode message C in the following way: The first 12 bytes are the initialization vector IV, and the last 16 bytes are the authentication tag T. The bytes in between are the ciphertext C.
2. Decrypt ciphertext C as (M,T′) = AES-128-GCM(K, IV, C) without any additional authenticated data.
3. Compare authentication tags T and T′. Do not abort as soon as a mismatch is found, but report an error only after all bytes have been compared.
4. If T ≠ T′, abort and discard the ciphertext.

3.4.2.8. Random generation

Whenever this specification requires generation random values, a cryptographically secure DRBG must be used.

3.5. Software authentication

Refer to *Software Authentication Server Specification* on how to obtain software tokens. MFi Program members will have access to this specification when it becomes available.

- A software authentication token, along with its corresponding UUID, must be provisioned on the accessory through factory provisioning (at the time of accessory manufacturing and firmware flashing).
- The software authentication token and its UUID must be decoded using Base64 from the file provided by Apple's server and stored as raw data bytes on the accessory.
- The UUID associated with the software authentication token must be registered with Apple server as defined in the *Software Authentication Server Specification* after provisioning on the accessory.
- The software authentication token must be stored in secure storage on the accessory.
- The provisioned software authentication token must persist through factory reset.
- The provisioned software authentication token is for one-time use only. The software authentication token and corresponding UUID will be required during pairing as part of the Find My network pairing process. A new software authentication token will be provided to the accessory during pairing and must be stored by the accessory for future use. See Pairing and key management for more details.

3.6. Apple server public keys

Whenever this specification requires Apple server signature verification, the accessory must use the Apple server signature verification key (Q_A).

Whenever this specification requires encryption to Apple server, the accessory must use the Apple server encryption key (Q_E).

- Apple server public keys must be provisioned in the accessory through factory provisioning (at the time of accessory manufacturing and firmware flashing) with integrity protection.
- Apple server public keys must be stored in secure storage on the accessory and protected against tampering.

MFi Program members will have access to Apple server public keys.

3.7. Factory reset

A factory reset must delete all Find My network data except the following:

- Product ID
- Vendor ID
- Firmware version
- Serial number
- Software authentication token
- Software authentication UUID
- Apple server public keys
  - Signature verification key (Q_A)

- Encryption key (Q_E)

The accessory must reenter Find my network pairing mode when the user initiates it. See Pairing for details.

3.8. Power cycle

A user may power cycle an accessory for various reasons (for example, battery replacement or user restart). When an accessory is power cycled, it shall start in separated state with the current secondary key as the separated key. See After power cycle for advertisement details.

3.9. Firmware updates

Accessories must support firmware updates. See Firmware update for more details.

- All firmware images must be authenticated and verified by the accessory using a mechanism that guarantees the integrity of the image from the manufacturer.
- Updated firmware must complete MFi certification requirements before release.
- Accessories must not allow a firmware image to be downgraded after a successful firmware update.

4. Bluetooth Requirements

4.1. Overview

Bluetooth Low Energy (BTLE) is used as the wireless transport for all communication between Apple products and accessories.

4.2. Bluetooth advertising

The accessory should advertise the Find My network payload at the $T_{FMN\_ADV\_INTERVAL}$ interval in a connectable advertising ADV_IND PDU to match the BTLE scan duty cycles of the Apple device. The accessory may advertise other Advertising Data Type (AD Type) in other advertising events. The accessory shall continue advertising the Find My network payload until all owner devices are connected. See Set max connections for details.

4.3. Bluetooth connection

The accessory must support at least two simultaneous connections in a peripheral role.

The connection interval of the BTLE link between the Apple device and accessory depends on the type of user interaction. An Apple device typically selects a connection interval in multiples of 15 ms. The accessory shall support a connection interval that is a multiple of 15 ms. The Apple device may use 990 ms as the BTLE connection interval to the accessory.

4.4. Bluetooth host

4.4.1. Services

The Find My network service must be instantiated as a primary service. The accessory must also support the following services:

- Tx Power service
- Battery service
- Bond management service
- Device information service

The accessory shall expose the Firmware Revision String and the PnP ID characteristics of the Device Information service. See Firmware version for more details. The Apple device requires the vendor ID, product ID, and firmware version for Find My network pairing.

4.4.2. MTU size

The accessory shall select a MTU size that is equal to or greater than the MTU request from the Apple device.

4.4.3. Link encryption key

The accessory is paired to the Apple device using the Find My network pairing protocol rather than the traditional Bluetooth pairing. To encrypt the BTLE link on every connection, the accessory must use the LTK generated by the Find My network protocol. See LTK generation for details on LTK generation and use.

4.4.4. Handling concurrent operations

An app on the Apple device may interact with the accessory over GATT or, if supported, connection-oriented L2CAP channels. Apple devices may connect and perform Find My network GATT operations independently from other interactions with the accessory.

The accessory shall support Find My network GATT interactions while simultaneously supporting GATT and connection-oriented L2CAP channels from other Apple devices.

4.4.5.Bluetooth pairing

Standard BTLE pairing is not required for accessories. Pairing is achieved at the Find My network protocol layer. See Pairing for more details.

4.4.6.Error codes

A list of error codes and their descriptions will be provided in an updated developer preview.

4.4.7. Time-out

Unless otherwise specified, the accessory must respond to all control point commands within 30 seconds.

4.5. Find My network service

4.5.1. Service

The Find My network service UUID is `32EEA8D0–F5A1–41B6–BFDB–97CD5FC926DB`.

This UUID will be updated in an updated developer preview.

4.5.2.Byte transmission order

All characteristics used with this service shall be transmitted with the least significant octet first (that is, little endian).

4.5.3.Characteristics

The UUID for Find My network service characteristics is 4F86XXXX–943B–49EF–BED4–2F730304427A, where XXXX is unique for each characteristic.

Table 4-1 Characteristics

Characteristic name	UUID	Requirement	Mandatory properties	Security permissions
Pairing Control Point	0x0001	Mandatory	Write, Indicate	Authorization Not Required
Configuration Control Point	0x0002	Mandatory	Write, Indicate	Authorization Required
Non Owner Control Point	0x0003	Mandatory	Write, Indicate	Authorization Not Required
Debug Control Point	0x0004	Mandatory	Write, Indicate	Authorization Required

A client characteristic configuration descriptor shall be included for all the characteristics, as required.

4.5.3.1.Pairing control point

The pairing control point enables you to pair an accessory with an owner device. The opCodes for the control point is defined in Table 4-2.

Table 4-2 Pairing control point opcodes

OpCode	OpCode value	Operands	GATT subprocedure	Direction
Initiate_pairing	0x100	SessionNonce E1	Write	To accessory
Send_pairing_data	0x101	OpCode E2	Indications	From accessory
Finalize_pairing	0x102	C2 E3 SeedS S2	Write	To accessory

OpCode	OpCode value	Operands	GATT subprocedure	Direction
Send_pairing_status	0x103	C3 Status OpCode E4	Indications	From accessory
Pairing_complete	0x104	None	Write	To accessory

4.5.3.2. Pairing control point procedures

The accessory, as server, shall indicate the pairing control point for responding to the commands from the Apple device.

4.5.3.2.1.Initiate pairing

The `Initiate_pairing` opcode is used to start the pairing session of an accessory from an Apple device.

Table 4-3 Initiate pairing

Operand	Data type	Size (octets)	Description
SessionNonce	bytes	32	Nonce generated by owner device
E1	bytes	89	Encrypted blob generated by owner device

4.5.3.2.2.Send pairing data

The `Send_pairing_data` opcode must be used by the accessory to respond to a pairing session request. The accessory must respond in 60 seconds.

Table 4-4 Send pairing data

Operand	Data type	Size (octets)	Description
OpCode	bytes	4	Context, value “Pair”
E2	bytes	1342	Encrypted blob generated by accessory

See Send pairing data for E2 generation details.

4.5.3.2.3.Finalize pairing

The `Finalize_pairing` opcode is used by an Apple device to confirm pairing. See Validate and confirm pairing for more details.

Table 4-5 Finalize pairing

Operand	Data type	Size (octets)	Description
C2	bytes	89	Shared commitment generated by Apple device
E3	bytes	1052	Encrypted software token that's vended by Apple server for each accessory
SeedS	bytes	32	Unique server seed for each accessory that's paired
S2	bytes	100	Apple server signature to confirm pairing

4.5.3.2.4.Send pairing status

The `Send_pairing_status` opcode must be used by the accessory to confirm pairing.

Table 4-6 Send pairing status

Operand	Data type	Size (octets)	Description
C3	bytes	60	Final commitment generated by accessory. See Collaborative key generation for C3 details.
Status	bytes	4	Success/failure status
opCode	bytes	4	Context, value "Ack"
E4	bytes	1266	Encrypted blob generated by accessory

See Send pairing status for E4 generation details.

4.5.3.2.5.Pairing complete

The `Pairing_complete` opcode is used to complete pairing the accessory from the Apple device. This opcode has no parameters.

4.5.3.3. Configuration control point

The configuration control point enables you to configure Find My network functionality on the accessory and enable Find My network interactions. The opCodes for the control point are defined in Table 4-7.

Table 4-7 Configuration control point opcodes

OpCode	OpCode Value	Operands	GATT subprocedure	Direction
Sound_Start	0x200	None	Write	To accessory
Sound_Stop	0x201	None	Write	To accessory
Persistent_ Connection_Status	0x202	Persistent ConnectionStatus	Write	To accessory
Set_Nearby_Timeout	0x203	NearbyTimeOut	Write	To accessory
Unpair	0x204	None	Write	To accessory
Configure_Separated_S-tate	0x205	NextKeyRoll NextSeparatedKeyOffset SeparatedKeyInterval	Write	To accessory
Latch_Separated_Key	0x206	None	Write	To accessory
Set_Max_Connections	0x207	MaxConnections	Write	To accessory
Set_UTC	0x208	CurrentTime	Write	To accessory
Get_Multi_Status	0x209	None	Write	To accessory
Get_Firmware_Version	0x210	None	Write	To accessory
Get_Battery_Status	0x211	None	Write	To accessory
Keyroll_Indication	0x212	KeyIndex	Indications	From accessory
Command_Response	0x213	CommandOpCode ResponseStatus	Indications	From accessory
Get_Multi_Status_Re-sponse	0x214	MultiStatus	Indications	From accessory
Get_Firmware_Ver-sion_Response	0x215	FirmwareVersion	Indications	From accessory
Get_Battery_Status_Re-sponse	0x216	BatteryStatus	Indications	From accessory

4.5.3.4.Configuration control point procedures

The paired Apple device initiates a configuration procedure using one of the messages defined in Table 4-7. The accessory must respond to the configuration procedure using the `Command_Response` message with an appropriate status code (see Table 4-14).

The accessory, as server, shall indicate the Configuration Control Point for responding to the commands from the Apple device.

4.5.3.4.1.Play sound—owner control point

Play sound requirements are applicable only for accessories that include a sound maker. See Product-specific requirements.

The `Sound_Start` opcode is used to play sound on the sound maker of the accessory.

The `Sound_Stop` opcode is used to stop an ongoing sound request.

If the sound event is completed or was not initiated by the Apple device, the accessory responds with `No_Sound_Session ResponseStatus` code.

4.5.3.4.2.Persistent connection status

The `Persistent_Connection_Status` opcode is used by the owner device to indicate whether the accessory is persistently connected using an always-connected BTLE link.

Table 4-8 Persistent connection status

Operand	Data type	Size (octets)	Description
Persistent_ Connection_Status	Boolean	1	0: Persistent connection disabled 1: Persistent connection enabled

4.5.3.4.3.Set nearby timeout

The `Set_Nearby_Timeout` opcode is used by the owner device to set the time duration to transition from nearby state to separated state, T_{NEARBY}. The valid range for the NearbyTimeOut parameter is 0 to 900 seconds. A NearbyTimeOut of 0 seconds indicates an immediate transition to separated state.

.

Table 4-9 Set NearbyTimeOut

Operand	Data type	Size (octets)	Description
NearbyTimeOut	Uint16	2	TimeOut in seconds

4.5.3.4.4. Unpair

The `Unpair` opcode is used to unpair the accessory from the Apple device. This opcode has no parameters. See Unpair for details.

4.5.3.4.5. Configure separated state

The `Configure_Separated_State` opcode is sent on every connection and indicates the time in milliseconds until the next 4 a.m. local time. The valid range for `nextPrimaryKeyRoll` parameter is 0 to 15 minutes.

The valid range for `secondaryKeyEvaluationIndex` parameter is `CurrentPrimaryKeyIndex` - 4 to `currentPrimaryKeyIndex` + 96.

Table 4-10 Configure separated state

Operand	Data type	Size (octets)	Description
NextPrimaryKeyRoll	Uint32	4	Time in milliseconds until the next primary key rotation
SecondaryKeyEvaluation-Index	Uint32	4	Primary key index at which secondary key is re-evaulated

4.5.3.4.6.Latch separated key

The `Latch_Separated_Key` opCode instructs the accessory to use the current primary key as Separated key until the next 4 a.m. local time. This message has no parameters.

4.5.3.4.7.Set max connections

The `Set_Max_Connections` opcode is used to set the maximum number of Bluetooth connections that must be supported by the accessory. Accessories shall support up to two simultaneous connections.

Table 4-11 Set max connections

Operand	Data type	Size (octets)	Description
MaxConnections	Uint8	1	Maximum Bluetoothconnected to be supported by accessory

4.5.3.4.8.Set UTC

The `Set_UTC` opcode is used to set the current UTC time on the accessory.

Table 4-12 Set UTC

Operand	Data type	Size (octets)	Description
CurrentTime	Uint32	4	Current UTC time in ms (Jan 1 2001 Epoch)

4.5.3.4.9.Keyroll indication

The `Keyroll_Indication` opcode must be used by the accessory to indicate that a primary key roll has occurred.

Table 4-13 Keyroll indication

Operand	Data type	Size (octets)	Description
KeyIndex	Uint8	6	Current primary key index

4.5.3.4.10.Command response

The `command_response` opcode must be used by the accessory to respond to every configuration procedure initiated by the Apple device. The `CommandOpCode` indicates the procedure that the accessory is responding to, and `ResponseStatus` indicates the status of the response.

Table 4-14 Command response

Operand	Data type	Size (octets)	Description
CommandOpCode	Uint16	2	The control procedure matching this response
ResponseStatus	Uint16	2	0x00 success 0x01 No_Sound_Session 0xFF Invalid_command

4.5.3.4.11.Get multi status response

The `Get_Multi_Status_Response` opcode must be used by the accessory to respond to the `Get_Multi_Status` command from the Apple device. The multiStatus is a bit mask that indicates the current state of the accessory. Setting a bit in `MultiStatus` indicates that the accessory is in that state.

Table 4-15 Multistatus

Operand	Data type	Size (octets)	Description
MultiStatus	Uint8	1	Bit0: MULTI_STATUS_PERSISTENT_CONNECTION, Bit1: RESERVED, Bit2: MULTI_STATUS_PLAYING_SOUND Bit13 MULTI_STATUS_DOWNLOADING Bit4: RESERVED Bit5: MULTI_STATUS_OWNER_CONNECTED Bit6: RESERVED Bit7: RESERVED

4.5.3.4.12. Firmware version

The `Get_Firmware_Version` and `Get_Firmware_Version_Response` opcode are used to exchange the firmware version of the accessory. The firmware revision string shall meet the requirements listed below.

For revision string x[.y[.z]] (for example, "100.1.1"):

- <x> is the major version number, required.
- <y> is the minor version number, required if it is non zero or if <z> is present.
- <z> is the revision version number, required if non zero.

The firmware revision must follow these rules:

- <x> is incremented when there is significant change; for example, 1.0.0, 2.0.0, 3.0.0, and so on.
- <y> is incremented when minor changes are introduced, such as 1.1.0, 2.1.0, 3.1.0, and so on.
- <z> is incremented when bug fixes are introduced, such as 1.0.1, 2.0.1, 3.0.1, and so on.
- Subsequent firmware updates can have a lower <y> version only if <x> is incremented.
- Subsequent firmware updates can have a lower <z> version only if <x> or <y> is incremented.
- Each number (major, minor, and revision version) must not be greater than (2^32 -1).
- The characteristic value must change after every firmware update.
- The `Get_Firmware_version_response` shall return the firmware version.

Table 4-16 Firmware version

Operand	Data type	Size (octets)	Description
FirmwareVersion	String	64 (maximum)	Firmware version

4.5.3.4.13. Battery status

The `Get_Battery_Status` and `Get_Battery_Status_Response` opcode are used to exchange the battery status of the accessory.

Table 4-17 Battery status

Operand	Data type	Size (octets)	Description
BatteryStatus	Uint8	1	0 = Full 1 = Medium 2 = Low 3 = Critically low

4.5.3.5. Non-owner control point

The non-owner control point enables a non-owner device to locate the accessory by playing a sound. The opCodes for the control point are defined in Table 4-18.

Table 4-18 Non-owner control point

OpCode	OpCode value	Operands	GATT subprocedure	Direction
Sound_Start	0x300	None	Write	To accessory
Sound_Stop	0x301	None	Write	To accessory
Command Response	0x302	CommandOpCode ResponseStatus	Indications	From accessory

This control point shall be available to the Apple device only when the accessory is in separated state. In all other states, the accessory shall return the `Invalid_command` error as the responseStatus in CommandResponse.

4.5.3.6.Non-owner control point procedures

The accessory, as server, shall indicate the non-owner control point for responding to the commands from the Apple device.

4.5.3.6.1. Play sound—non-owner control point

Play sound requirements are applicable only to accessories that include a sound maker. See Product-specific requirements.

The `Sound_Start` opcode is used to play sound on the sound maker of the accessory.

The `Sound_Stop` opcode is used to stop an ongoing sound request.

If the sound event is completed or was not initiated by the Apple device, the accessory responds with the `No_Sound_Session` `ResponseStatus` code.

4.5.3.7.Debug control point

The debug control point enables you to debug the accessory during development. This control point shall not be enabled in shipping firmware.

The opCodes for the control point are defined in Table 4-19.

Table 4-19 Debug control point

OpCode	OpCode value	Operands	GATT subprocedure	Direction
Set_Key_Rotation_Timeout	0x400	Timeout	Write	To accessory
Retrieve_Logs	0x401	None	Write	To accessory
Log_Response	0x402	LogResponse	Indications	From accessory

4.5.3.8.Debug control point procedures

4.5.3.8.1.Set key rotation time-out

The `Set_Key_Rotation_Timeout` debug command accelerates key rotation by configuring a short time-out.

Table 4-20 Set key rotation timeout

Operand	Data type	Size (octets)	Description
Timeout	Uint32	4	Time in milliseconds until the next primary key rotation

4.5.3.8.2.Retrieve logs

The `Retrieve_Logs` debug command is used to dump logs from an accessory. The accessory transfers the logs with multiple `Log_Response` Indications. The size of the `Log_Response` is limited by the MTU size negotiated during connection setup. The accessory indicates the end of the log dump by sending a `Log_Response` with empty payload.

5. Advertisements

5.1. BTLE advertising

An accessory that is not Find My network paired shall advertise the Find My network service as a primary service when the user puts the accessory in pairing mode.

After Find My network pairing, the accessory shall advertise the Find My network BTLE payload in the format defined in Table 5-1.

Table 5-1 BTLE advertising

AdvAddress	Manuf AD Type		Find My network payload
PrimaryKey[0..5]	AD Type	CompanyID	Nearby or separated payload

The Find My network advertising payloads replaces the `AdvA` field of the advertising PDU defined by the BT SIG with the first `0 to 5` bytes of the current key. The nearby or separated state of the accessory determines the current key. Most significant bits of byte `0` shall be `0b11`, indicating a static device address.

The Find My network advertisement payload shall not contain other data types. An accessory must always advertise the Find My network payloads once every T_{ADVINT}. The accessory may use another advertising instance to broadcast other data types and services.

The manufacturer AD type is defined by the BT SIG, and the payload indicates that the type is Apple.

Table 5-2 Manufacturer data

Byte	Value	Description
0	3	Length of manufacturer AD type
1	0xFF	Manufacturer data AD type
2..3	0x004C	Apple company ID

5.1.1. Payload for nearby state

When the accessory is in the nearby state or connected to a paired owner device, the advertising payload format must be as defined in Table 5-3.

Table 5-3 Payload for nearby state

Byte	Value	Description
0	0x12	Apple payload type

Byte	Value	Description
1	0x02	Length of payload
2	Bits 0–1: Reserved. Bit 2: Maintained Bits 3–4: Reserved Bits 5: 0b1 Bits 6–7: Battery state.	Maintained Set if owner connected within current key rotation period (15 minutes) Battery state definition 0 = Full 1 = Medium 2 = Low 3 = Critically low
3	Bits 0–1: Public key Bits 2–7: Reserved	Bits 6–7 of byte 0 of the primary key (P_i)

5.1.2. Payload for separated state

When the accessory is in the separated state, the advertising payload format must be as defined in Table 5-4.

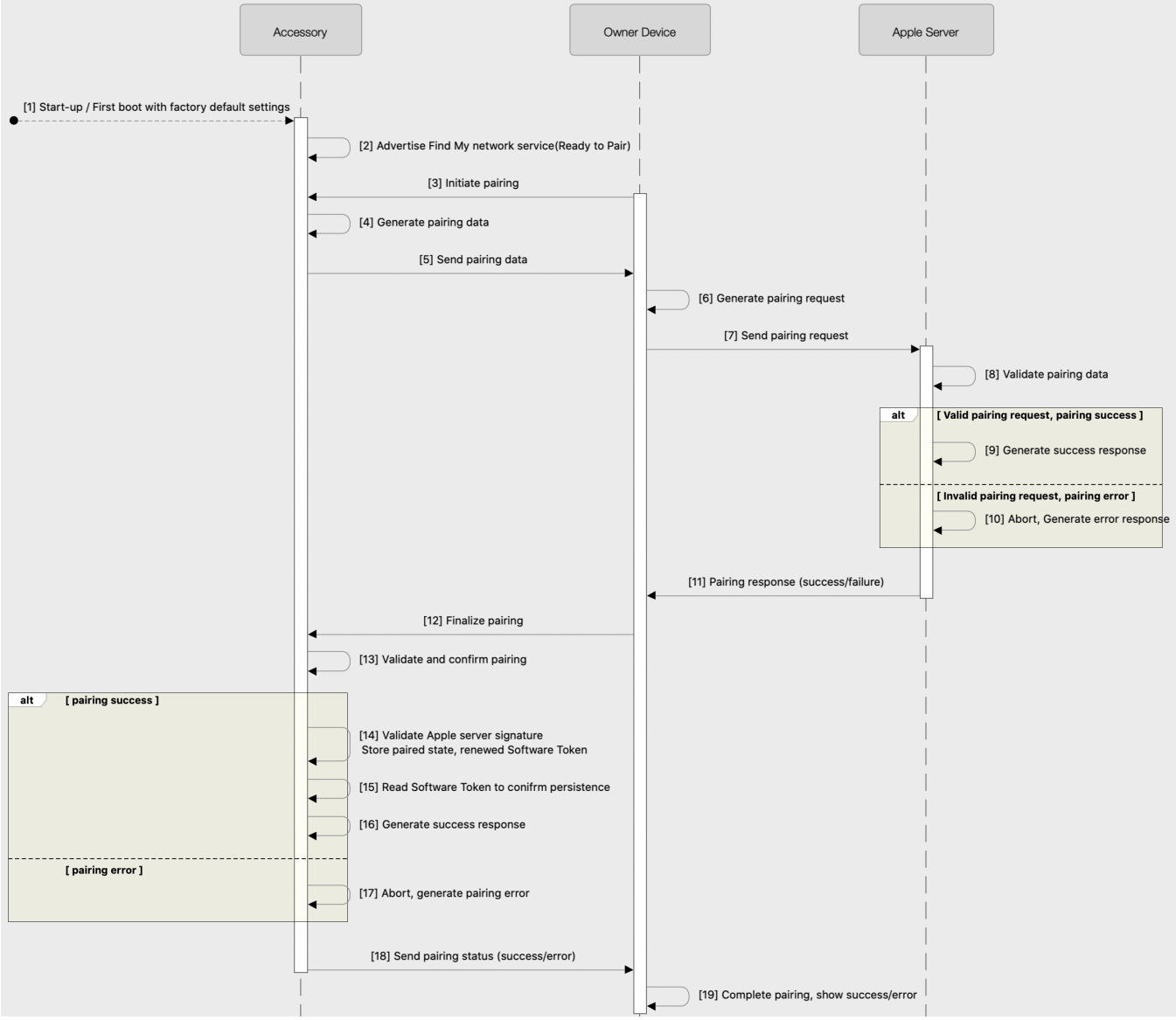
Table 5-4 Payload for wild state

Byte	Value	Description
0	0x12	Apple payload type
1	0x19	Length of payload
2	Bits 0–1: Reserved. Bit 2: Maintained Bits 3–4: Reserved Bits 5: 0b1 Bits 6–7: Battery state.	Maintained Set if owner connected within current key rotation period (15 minutes) 0 = Full 1 = Medium 2 = Low 3 = Critically low
3 — 24	Separated public key	Bytes 6–27 of the Public Key, P_i or PW_j depending on accessory state. See Nearby to separated, Separated to separated, and After power cycle for possible separated state transitions.
25	Bits 0–1: Public key Bits 2–7: Reserved	Bits 6–7 of byte 0 of the public key (P_i or PW_j)
26	Hint	Byte 5 of the Bluetooth address of the current primary key P_i

6. Pairing and Key Management

6.1. Overview

An accessory must be paired to an owner device before it can be locatable. An owner device will initiate the standard BTLE encryption before it accesses the Find My network services.



6.2. Pairing

Find My network pairing is initiated by the owner device using the pairing control point procedures. When an accessory pairs, it must not expose the Find My network pairing control point and it must respond to any of the pairing control point procedures with an `invalid_command` error message.

An accessory will not be able to Find My network pair if it is paired to an owner device with a different Apple ID.

6.2.1. Pairing mode

The accessory must require explicit user intent to enable the Find My network pairing mode. When the user initiates the Find My network pairing mode, the accessory must advertise the Find My network service as a primary service. See Find My network service. The accessory must exit the pairing mode after a time-out.

6.2.2.Generate pairing data

Upon establishing standard BLE encrypted pairing session, the accessory must generate collaborative commitment (C1) to start the pairing process and generate per pairing session encryption key seed (SeedK1). See Random generation for the generation of SeedK1. The accessory must regenerate SeedK1 for every new pairing session.

See Collaborative key generation for C1 details.

See Send pairing data pairing control point for details.

6.2.3.Send pairing data

The accessory must send encrypted payload generated using Apple server encryption key (`Q_E`).

The parameters listed in Table 6-1 are included in generating E2. See ECIES Encryption for E2 generation.

Table 6-1 Payload to generate E2

Key	Data type	Size (octets)	Description
SessionNonce	bytes	32	Nonce generated by Apple device
C1	bytes	32	Data sent by the accessory as initial commitment for pairing (see Collaborative key generation for C1 details)
Software auth token	bytes	1024	Software authentication token that's vended by Apple for each accessory
Software auth UUID	bytes	16	Accessory UUID that's associated with software auth

Key	Data type	Size (octets)	Description
Serial Number	bytes	16	Accessory serial number
Product ID	String	16	Accessory product ID
Vendor ID	String	16	Accessory vendor ID
FW version	String	4	Accessory firmware version
E1	bytes	89	Encrypted blob generated by owner device
SeedK1	bytes	16	Per pairing session seed for encryption key

6.2.4.Finalize pairing

The owner device initiates the finalize pairing process to complete pairing. See Finalize pairing for details.

6.2.5.Validate and confirm pairing

The accessory must validate the Apple server signature (S2) using an Apple server signature verification key (Q_A) in order to finalize pairing.

The parameters listed in Table 6-2 are included in generating S2.

Table 6-2 Payload to generate signature message for S2 verification

Key	Data type	Size (octets)	Description
Software auth UUID	bytes	16	Accessory UUID that's associated with software token
SessionNonce	bytes	32	Nonce generated by owner device
SeedS	bytes	32	Unique server seed for each accessory that's paired
H1	bytes	32	Compute H1=SHA-256(C2)
E1	bytes	89	Encrypted blob generated by owner device
E3	bytes	1052	Encrypted software token that's vended by Apple server for each accessory

See Apple server public keys and ECDSA signature verification for signature verification key (Q_A) details.

In case of signature verification failure, the accessory must abort pairing. See Send Pairing Status for more details about success and error status.

If Apple server signature verification is successful, then the accessory must decrypt Apple server encrypted blob (E3) using per pairing session symmetric AES 128-bit key (K1).

See derivation of the Pairing Session Key K1 for details on obtaining K1. See AES-GCM decryption for E3 decryption details.

If S2 verification and E3 decryption are successful, then the accessory must store a new software token from E3 and generate a collaborative key (C3) as an acknowledgement to confirm pairing.

The accessory must always use the latest (renewed) software token for any subsequent operations that require authentication with Apple servers (for example, unpair).

See Collaborative key generation for C3 details. See Finalize pairing for E3 details.

6.2.6.Send pairing status

After successful pairing, the accessory must go into nearby state and send an acknowledgement to the owner device to confirm the pairing.

The accessory must initialize a 64-bit counter to 0. This counter is used along with the serial number in the NFC payload.

In case of pairing error, the accessory must abort pairing and send a pairing error code. For both success and error, the accessory must generate an encrypted blob (E4) and send it to the owner device.

The payload parameters listed in Table 6-3 are included in generating E4. See ECIES Encryption for E4 generation.

Table 6-3 Payload to generate E4

Key	Data type	Size (octets)	Description
Software auth UUID	bytes	16	Accessory UUID that's associated with software token
Serial Number	bytes	16	Accessory serial number
SessionNonce	bytes	32	Nonce generated by the owner device
E1	bytes	89	Encrypted blob generated by the owner device
Software token	bytes	1024	Latest Software token
Status	bytes	4	Success/failure status code
OpCode	bytes	4	Context, value ="Ack"

Pairing error codes will be provided in an updated developer preview. See Send pairing status for details.

6.3. Key management

6.3.1. Key definitions

As part of a successful pairing flow, the accessory and the owner device will collaboratively generate both of the following:

- A master public key, P
- Two symmetric keys, SKN and SKS

A derivative of the public key P will be broadcast over BTLE. Finder devices can use it to encrypt their current location and provide it to Apple servers for the accessory owner to download and decrypt.

Additionally, the accessory and the server generate a shared secret. The shared secret is used to derive a key and protects requests related to obtaining lost mode information:

- Secret shared with server: ServerSharedSecret
- Symmetric key for pairing session: K1
- Symmetric key for queries with serial number: KSN

6.3.2. Key sequences and rotation policy

The accessory must generate public key sequences with different key rotation intervals, referred to as primary and secondary keys.

- P and SKN are used to derive the primary key (P_i), which rotates every 15 minutes.
- P and SKS are used to derive the secondary key (PW_j), which rotates every 24 hours (that is, after every 96 iterations of primary key P_i).

6.3.3. Bluetooth advertisement key selection policy

6.3.3.1. After pairing

The accessory must use the primary key P_i (where $i=1$) as a BTLE advertisement and enters nearby state. See Payload for nearby state for details.

6.3.3.2. Nearby to nearby state transition

If at the end of period 'i' the accessory is still in nearby state, it must use the next primary key P_{i+1} (where 'i' is the last primary key index) as a BTLE advertisement. See Payload for nearby state for details.

6.3.3.3. Nearby to separated state transition

When the accessory switches to separated state, it must continue to use the current primary key P_i as a BTLE advertisement until the end of the current separated key period (4 a.m. local time). See Payload for separated state for details.

6.3.3.4. Separated to separated state transition

If at the end of the current separated key period (4 a.m. local time) the accessory is still in separated state, and it was previously advertising the last primary key P_i right after the state transition, it must compute `j=i/96+1` and the secondary key PW_j and use the latter as a BTLE advertisement.

If at the end of the current separated key period (4 a.m. local time) the accessory is still in separated state, and it was previously advertising the secondary key PW_j, it now must use the next secondary key PW_{j+1} as a BTLE advertisement. See Payload for separated state for details.

6.3.3.5. After power cycle

The accessory must compute `j=i/96+1` and the secondary key PW_j (where 'i' is the current primary key index) and use the latter as a BTLE advertisement. See Payload for separated state for details.

6.3.4.Key schedule definitions

a || b denotes concatenation of the values a and b.

G is the base point of the NIST P-224 elliptic curve. See FIPS 186-4, D.1.2.2. Curve P-224.

q is the order of the base point G. x(P) denotes the x coordinate of the elliptic curve point P.

ANSI-X9.63-KDF(Z, sharedInfo) denotes the KDF described by SEC1, 3.6.1 ANSI X9.63 Key Derivation Function. Z is the secret value (the input key material) and sharedInfo is data shared between the two parties.

Random values and scalars must be generated using a cryptographically secure DRBG. See Operations.

6.3.4.1. Collaborative key generation

As part of the pairing flow, the owner device and the accessory must collaboratively generate a public key P and two symmetric keys, SKN and SKS.

1. The accessory generates a P-224 scalar s (see Random scalar generation) and a 32-byte random value r. It sends the value `C1 = SHA-256(s || r), where len(C1) = 32 bytes,` to the owner device. (See Send pairing data.)
2. The owner device generates a P-224 scalar s′ (see Radom Scalar Generation) and a 32-byte random value r′. It computes $S' = s' \cdot G$ and sends `C2 = {S', r'}, where len(C2) = 89 bytes,` to the accessory. (See Finalize pairing.)
3. The accessory checks S′ and aborts if it is not a valid point on the curve. (See Elliptic curve point validation.) It computes the final public key $P = S' + s \cdot G$ and sends `C3 = {s, r}, where len(C3) = 60 bytes,` to the owner device. (See Send pairing status.)
4. The owner device aborts if s is not a valid P-224 scalar (see Scalar validation) or if $C1 \neq SHA\text{-}256(s \,||\, r)$. It computes the final public key $P = S' + s \cdot G$ and the private key $d = s + s' \pmod q$.
5. Both the owner device and the accessory compute the final symmetric keys SKN and SKS as the 64-byte output of ANSI-X9.63-KDF(x(P), r || r′), where SKN is the first 32 bytes and SKS is the last 32 bytes.

6.3.4.2. Derivation of primary and secondary keys

The accessory must derive primary and secondary keys from the public key P generated at pairing time. P itself must never be sent out and must be stored in a secure location.

For a given 15-minute period i:

1. Derive $SKN_i = ANSI\text{–}X9.63\text{–}KDF(SKN_{i-1}, \text{“update”})$, where SKN_0 is the SKN as agreed upon at pairing time.
2. Derive $AT_i = (u_i, v_i) = ANSI\text{-}X9.63\text{-}KDF(SKN_i, \text{“diversify”})$ where $len(AT_i) = 72$ bytes and $len(u_i) = len(v_i) = 36$ bytes.
3. Reduce the 36-byte values u_i, v_i into valid P-224 scalars by computing the following:

   a. $u_i = u_i \pmod{q-1} + 1$

   b. $v_i = v_i \pmod{q-1} + 1$
4. Compute $P_i = u_i \cdot P + v_i \cdot G$.

Secondary keys are generated as shown above, using period j instead of i and SKS instead of SKN. The result will then be called PW_j instead of P_i.

6.3.4.3. Derivation of link encryption key LTK_i

The Find My network key generation algorithm generates LTKs, rotating every 15 minutes. The accessory shall use the LTK that corresponds to the current key period as the LTK to encrypt the link on connection to the owner device. A paired owner device also picks the same LTK to encrypt the link. If the device is not a paired Apple device or if the LTK results in a failed encryption, the accessory must disconnect.

The accessory must derive a new link encryption key LTK_i for every 15-minute period i. If the paired owner device is nearby, it can use this key to establish a Bluetooth connection and encrypt the link.

For a given 15-minute period i:

1. Derive the symmetric key $SKN_i = ANSI\text{–}X9.63\text{–}KDF(SKN_{i-1}, \text{“update”})$, where SKN_0 is the symmetric key SKN as agreed upon at pairing time.
2. Derive the Intermediate key $IK_i = ANSI\text{–}X9.63\text{–}KDF(SKN_i, \text{“intermediate”})$, where $len(IK_i) = 32$ bytes.
3. Derive the Link Encryption key $LTK_i = ANSI\text{–}X9.63\text{–}KDF(IK_i, \text{“connect”})$, where $len(LTK_i) = 16$ bytes.

6.3.4.4. Derivation of command key CK_i

The accessory must derive a new command key CK_i for every 15-minute period i. The paired owner device uses CK_i to ensure the authenticity of commands sent to the accessory.

For a given 15-minute period i:

1. Derive the symmetric key $SKN_i = ANSI\text{–}X9.63\text{–}KDF(SKN_{i-1}, \text{“update”})$, where SKN_0 is the symmetric key SKN as agreed upon at pairing time.
2. Derive the Intermediate key $IK_i = ANSI\text{–}X9.63\text{–}KDF(SKN_i, \text{“intermediate”})$, where $len(IK_i) = 32$ bytes.

3. Derive the command key CK_i = ANSI–X9.63–KDF(IK_i, "command"), where len(CK_i) = 32 bytes.

6.3.4.5. Derivation of the NearbyAuthToken$_i$

The accessory and owner device will derive a new $NearbyAuthToken_i$ for a given 15-minute period i. The paired owner device broadcasts with an advertising address derived from the $NearbyAuthToken_i$. An accessory in separated state must switch to nearby state upon detecting such a broadcast.

For a given 15-minute period i:

1. Derive the primary key P_i as shown in Derivation of primary and secondary keys.
2. Derive the command key CK_i as shown in Derivation of command key CKi.
3. Denote $x(P_i)$ as the x-coordinate of the primary key P_i, where $x(P_i)$ is represented as a 28-byte big-endian integer.
4. Compute $NOAT_i$ = HMAC–SHA256(CK_i, $x(P_i)$ || "NearbyAuthToken").
5. Compute $NearbyAuthToken_i$ = MostSignificant6Bytes($NOAT_i$).

6.3.4.6. Derivation of ServerSharedSecret

Upon successful pairing, the accessory must generate and retain ServerSharedSecret, where ServerSharedSecret is a 32-byte shared secret:

ServerSharedSecret = ANSI–X9.63–KDF(SeedS || SeedK1, "ServerSharedSecret")

6.3.4.7. Derivation of the pairing session key K1

To generate the NFC tap payload, KSN must be generated as follows, where K1 is a 16-byte symmetric key:

K1 = ANSI–X9.63–KDF(ServerSharedSecret, "PairingSession")

6.3.4.8.Derivation of the serial number protection key

To generate the NFC tap payload, KSN must be generated as follows, where KSN is a 16-byte symmetric key:

KSN = ANSI–X9.63–KDF(ServerSharedSecret, "SerialNumberProtection")

6.4. Unpair

Unpair action is initiated by the paired owner device to delete Find My network data.

See Unpair for the unpair procedure. See Factory reset for details on resetting the accessory.

7. Unwanted Tracking Detection

7.1. Overview

During separated UT state, sound playback from the accessory is designed to bring awareness to the person with whom it's detected. Accessories that support motion-triggered UT sound alerts (see Product-specific requirements) must implement the requirements from this chapter.

7.2. Hardware

7.2.1. Motion detector

The accessory must include a motion detector that can detect accessory motion reliably (for example, an accelerometer). If the accessory includes an accelerometer, it must be configured to detect an orientation change of ±10° along any two axes of the accessory.

7.2.2. Sound maker

The accessory must include a sound maker (for example, a speaker) to play sound when motion is detected in separated UT state.

It must also play sound when a non-owner tries to locate the accessory by initiating a play sound command from a non-owner device when the accessory is in range and connectable through BTLE. See Play sound—non-owner control point.

Additional requirements on acceptable sound maker performance (for example, minimum decibel level) will be provided in an updated developer preview.

7.3. Implementation

After $T_{SEPARATED\_UT\_TIMEOUT}$ in separated state, the accessory must enter the separated UT state and enable the motion detector (for example, accelerometer) to detect any motion within $T_{SEPARATED\_UT\_SAMPLING\_RATE1}$.

If motion is not detected within the $T_{SEPARATED\_UT\_SAMPLING\_RATE1}$ period, the accessory must stay in this state until it exits separated state.

If motion is detected within the $T_{SEPARATED\_UT\_SAMPLING\_RATE1}$ the accessory must play a sound. After first motion is detected, the movement detection period is decreased to $T_{SEPARATED\_UT\_SAMPLING\_RATE2}$. The accessory must continue to play a sound for every detected motion. The accessory shall disable the motion detector for $T_{SEPARATED\_UT\_BACKOFF}$ under either of the following conditions:

- Motion has been detected for 20 seconds at $T_{SEPARATED\_UT\_SAMPLING\_RATE2}$ periods.
- Ten sounds are played.

If the accessory is still in separated state at the end of $T_{SEPARATED\_UT\_BACKOFF}$, the UT behavior must restart.

A BTLE connection from a paired Apple device must reset the separated UT behavior and transition the accessory to connected state.

8. NFC Requirements

8.1. Overview

Accessories that include NFC (see Serial number lookup) must support the requirements from this chapter.

8.2. Hardware

These are the hardware requirements for accessories that include NFC:

- The accessory must use a programmable NFC tag.
- NFC tags must use the NFC Data Exchange Format (NDEF) as defined by *NFC Forum™ in NDEF 1.0 NFCForum-TS-NDEF 1.0*.
- An NDEF message is defined as a group of individual NDEF records as defined by *NFC Forum™ in NFC Record Type Definition (RTD) RTD 1.0 NFCForum-TS-RTD 1.0*.
- The Find My network payload for NFC tags must use NDEF URI Record Type Definition as defined by *NFC Forum™ in URI Record Type Definition RTD-URI 1.0 NFCForum-TS-RTD URI 1.0*.
- The minimum payload that must be supported is 30 bytes.
- NFC tag types must be type 2 or greater.
- The NFC tag should not be scannable when the Find My network-enabled accessory is still in the packaging.
- The Find My network payload must be scannable when holding the top of the iOS controller near the center of the NFC tag on the accessory. Recommended NFC tag performance guidelines are defined by *NFC Forum™ in Tag Performance Requirements Document*.

8.3. Implementation

On NFC tap, the accessory must generate encrypted payload using the Apple server encryption key (Q_E), including the serial number, a counter, and MAC, using the KSN symmetric key. The counter must monotonically increase every time an NFCTap occurs.

See ECIES Encryption for generating encrypted payload.

The NFC on an accessory must be configured as an NFC tag.

Unpaired accessories advertise the following payload:

```
https://found.apple.com/accessory?
pid=%04x&b=%02x&pt=%04x&fv=%08x&bt=%s&sr=%s
```

Paired accessories advertise the following payload:

```
https://found.apple.com/accessory?
pid=%04x&b=%02x&pt=%04x&fv=%08x&e=%s&op=%s
```

The payload parameters are defined in Table 8-1.

Table 8-1 NFC payload

Key	Size (Bytes)	NFC URL Format	Notes
b	1	ASCII Hex String	Battery status
bt	6	ASCII Hex String	Bluetooth MAC address
fv	4	ASCII Hex String	FW version, in little endian format
op	4	ASCII	Context, value “tap”
e	141	ASCII Hex String	Encrypted: Serial Number || Counter || HMAC(KSN, SerialNumber || Counter || op) || op
pid	2	ASCII Hex String	Accessory product ID
pt	2	ASCII Hex String	Accessory vendor ID
sr	16	ASCII	Accessory serial number

9. Timers and Constants

9.1. Overview

Table 9-1 defines the timers and constants used by the Find My network protocol.

Table 9-1 Timers and constants

Timer Name	Value	Description
$T_{SEPARATED\_UT\_TIMEOUT}$	3 days	Time span in separated state before enabling separated UT state.
$T_{SEPARATED\_UT\_BACKOFF}$	6 hours	Period to disable motion detector if accessory is in separated UT state.
T_{NEARBY}	15 minutes	Default value. Configured by the owner device on connection.
$T_{FMN\_ADV\_INTERVAL}$	2 seconds	Find My network BTLE Advertising Interval.
$T_{SEPARATED\_UT\_SAMPLING\_RATE1}$	10 seconds	Motion detector sampling rate when separated UT state is enabled.
$T_{SEPARATED\_UT\_SAMPLING\_RATE2}$	0.5 seconds	Motion detector sampling rate when movement is detected in separated UT state.

10. Firmware Update

10.1. Overview

The accessory must support the download of new firmware image from the owner device.

Details on firmware download procedures will be provided in an updated developer preview.

11. Revision History

Revision history

Version	Date	Notes
1.0	2020-06-22	Developer Preview 1