IMPLEMENTATION DRAFT

FIDO Bluetooth and Bluetooth Low Energy Protocol Specification v1.0

FIDO Alliance Implementation Draft 14 May 2015

This version:
https://fidoalliance.org/specs/fido-undefined-undefined-id-20150514/fido-u2f-bt-protocol-v1.0-undefined-id-20150514.html
Previous version:
https://fidoalliance.org/specs/fido-u2f-bt-protocol-v1.0-Member Submission-20140721.html
Editors:
Alexei Czeskis, Google, Inc.
Juan Lang, Google, Inc.
Contributors:
Scott Walsh, Plantronics, Inc.
Deniz Akkaya, Yubico, Inc.
Jakub Pawlowski, Google, Inc.
Hannes Tschofenig, ARM Ltd.

Abstract

The FIDO U2F framework was designed to be able to support multiple Authenticator form factors. This document describes the communication protocol with Authenticators over Bluetooth classic (BT) and Bluetooth Low Energy (BLE).

There are multiple form factors possible for Authenticators. Some might be low cost, low power devices, and others might be implemented as an additional feature of a more powerful device, such as a smartphone. The design proposed here is meant to support multiple form factors, including but not necessarily limited to these two examples.

Status of This Document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current FIDO Alliance publications and the latest revision of this technical report can be found in the FIDO Alliance specifications index at https://www.fidoalliance.org/specifications/.

This document was published by the FIDO Alliance as a Implementation Draft. This document is intended to become a FIDO Alliance Proposed Standard. If you wish to make comments regarding this document, please Contact Us. All comments are welcome.

Table of Contents

1. Notation
    - 1.1 Key Words
2. Pairing
3. Link Security
4. Framing
    - 4.1 Request from Client to Authenticator
    - 4.2 Response from Authenticator to Client
    - 4.3 Command and status constants
5. Bluetooth Classic
    - 5.1 Procedure Overview

5.2 Discovery Mode
5.3 Service Discovery Protocol
5.4 Communication
5.5 RFCOMM Framing
6. Bluetooth Low Energy
6.1 GATT Service Description
6.1.1 U2F Service
6.1.2 Device Information Service
6.1.3 Generic Access Service
6.2 Protocol Overview
6.3 Authenticator Advertising Format
6.4 Requests
6.5 Responses
6.6 Framing fragmentation
6.7 Implementation Considerations
6.7.1 Bluetooth pairing: Client considerations
6.7.2 Bluetooth pairing: Authenticator considerations
6.7.3 Handling command completion
6.7.4 Data throughput
6.7.5 Advertising
7. Bibliography
A. References
A.1 Normative references

1. Notation

Type names, attribute names and element names are written as `code`.

String literals are enclosed in “”, e.g. “UAF-TLV”.

In formulas we use “|” to denote byte wise concatenation operations.

DOM APIs are described using the ECMAScript [ECMA-262] bindings for WebIDL [WebIDL].

UAF specific terminology used in this document is defined in [FIDOGlossary].

1.1 Key Words

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in [RFC2119].

2. Pairing

BT and BLE are long-range wireless protocols and thus have several implications for privacy, security, and overall user-experience. Because they are wireless, BT and BLE may be subject to monitoring, injection, and other network-level attacks.

For these reasons, Clients and Authenticators MUST create and use a long-term link key (LTK) and SHALL encrypt all communications. Authenticator MUST never use short term keys.

Because BT and BLE have poor ranging (*i.e.,* there is no good indication of proximity), it may not be clear to a FIDO Client with which BT/BLE Authenticator it should communicate. Pairing is the only mechanism defined in this protocol to ensure that FIDO Clients are interacting with the expected BT/BLE Authenticator. As a result, Authenticator manufacturers SHOULD instruct users to avoid performing Bluetooth pairing in a public space such as a cafe, shop or train station.

One disadvantage of using standard Bluetooth pairing is that the pairing is "system-wide" on most operating systems. That is, if an Authenticator is paired to a FIDO Client which resides on an operating system where Bluetooth pairing is "system-wide", then any application on that device might be able to interact with an Authenticator. This issue is discussed further in Implementation Considerations.

3. Link Security

For BLE connections, the Authenticator SHALL enforce `Security Mode 1, Level 2` (unauthenticated pairing with encryption) before any U2F messages are exchanged.

For BT connections, the Authenticator SHOULD use Secure Simple Pairing when possible, `Security Mode 4` or better. Encryption SHALL be enabled using a key size of 16 bytes before any U2F messages are sent.

4. Framing

Conceptually, framing defines an encapsulation of U2F raw messages responsible for correct transmission of a single request and its response by the transport layer (Bluetooth Classic or Bluetooth Low Energy).

All requests and their responses are conceptually written as a single frame. The format of the requests and responses is given first as complete frames. Fragmentation is discussed next for each type of transport layer.

4.1 Request from Client to Authenticator

Request frames must have the following format

Offset	Length	Mnemonic	Description
0	1	`CMD`	Command identifier
1	1	`HLEN`	High part of data length

2	1	LLEN	Low part of data length
3	s	DATA	Data (s is equal to the length)

Supported commands are PING and MSG. The constant values for them are described below.

Data format is defined in [U2FRAWMESSAGES].

4.2 Response from Authenticator to Client

Response frames must have the following format, which share a similar format to the request frames:

Offset	Length	Mnemonic	Description
0	1	STAT	Response status
1	1	HLEN	High part of data length
2	1	LLEN	Low part of data length
3	s	DATA	Data (s is equal to the length)

When the status byte in the response is the same as the command byte in the request, the response is a successful response. The value ERROR indicates an error, and the response data contains an error code as a variable-length, big-endian integer. The constant value for ERROR is described below. The set of error values is TBD.

Note that the errors sent in this response are errors at the encapsulation layer, *e.g.,* indicating an incorrectly formatted request, or possibly an error communicating with the Authenticator's U2F message processing layer. Errors reported by the U2F message processing layer itself are considered a success from the encapsulation layer's point of view, and are reported as a complete MSG response.

Data format is defined in [U2FRAWMESSAGES]. Note that as per [U2FRAWMESSAGES] (and unlike the NFC transport specification), all communication SHALL be done using extended length APDU format.

4.3 Command and status constants

Constant	Value
PING	0x81
MSG	0x83
ERROR	0xbf

5. Bluetooth Classic

5.1 Procedure Overview

The general procedure is as follows:

1. If the Client and Authenticator are not yet bonded, the Authenticator becomes discoverable (enters Discoverable Mode). An Authenticator SHALL only allow connections from new Clients while in this mode.
2. Client connects to Authenticator. If not already paired, Client and Authenticator perform BT bonding to create a link key and connect. Authenticator SHALL only allow connections from previously bonded Clients without user intervention.
3. Client performs service discovery on the Authenticator.
4. Client connects to the FIDO U2F service.
5. Client writes a request (*e.g.*, an enroll request)
6. Authenticator evaluates the request and responds.
7. The connection is closed by the Client or the connection times out and is closed by the Authenticator.

5.2 Discovery Mode

When the Authenticator is in Bluetooth discovery mode, it SHOULD include a device name in the Extended Inquiry Response (EIR) packet. The device name should be distinctive and user-identifiable. For example, "ACME Key" would be an appropriate name, while "XJS4" would not be.

5.3 Service Discovery Protocol

The Authenticator SHALL contain a Service Discovery Protocol (SDP) record with the following data:

```
uint8 fido_client_spp_sdprecord [] =
{
  0x09, 0x00, 0x01, /* ServiceClassIDList(0x0001) */
  0x35,0x11, /* DataElSeq 17 bytes */
  0xT, 0xB, 0xD, 0xT,0xB, 0xD, 0xT, 0xB, 0xD, 0xT, 0xB, 0xT,
0xB , 0xD,0xT, 0xB, 0xD, /*UUID TBD*/
  0x09, 0x00, 0x04, /* ProtocolDescriptorList(0x0004) */
  0x35, 0x0c, /* DataElSeq 12 bytes */
  0x35, 0x03, /* DataElSeq 3 bytes */
  0x19, 0x01, 0x00, /* UUID L2CAP(0x0100) */
  0x35, 0x05, /* DataElSeq 5 bytes */
  0x19, 0x00, 0x03, /* UUID RFCOMM(0x0003) */
  0x08, 0x00,
/* uint8 0x00 - Change 0x00 to actual RFCOMM Channel Number */
  0x09, 0x00, 0x06, /* LanguageBaseAttributeIDList(0x0006) */
  0x35, 0x09, /* DataElSeq 9 bytes */
```

```
  0x09, 0x65, 0x6e, /* uint16 0x656e */
  0x09, 0x00, 0x6a, /* uint16 0x006a */
  0x09, 0x01, 0x00, /* uint16 0x0100 */
  0x09, 0x01, 0x00,
/* ServiceName(0x0100) = "U2FAUTHDEVICE" */
  0x25, 0x0D, /* String length 13 */
  'U','2','F','A','U','T','H','D','E','V','I','C','E'
};
```

5.4 Communication

If one or both of the Authenticator and Client only supports Classic Bluetooth, they SHALL communicate over RFCOMM.

If both Authenticator and Client are dual mode devices, they SHALL communicate using GATT over L2CAP on the BREDR connection.

5.5 RFCOMM Framing

No fragmentation is supported as communication over RFCOMM should be able to handle all messages without fragmentation.

6. Bluetooth Low Energy

Authenticator and Client devices using BLE SHALL conform to Bluetooth Core Specification 4.0 or later [BTCORE]

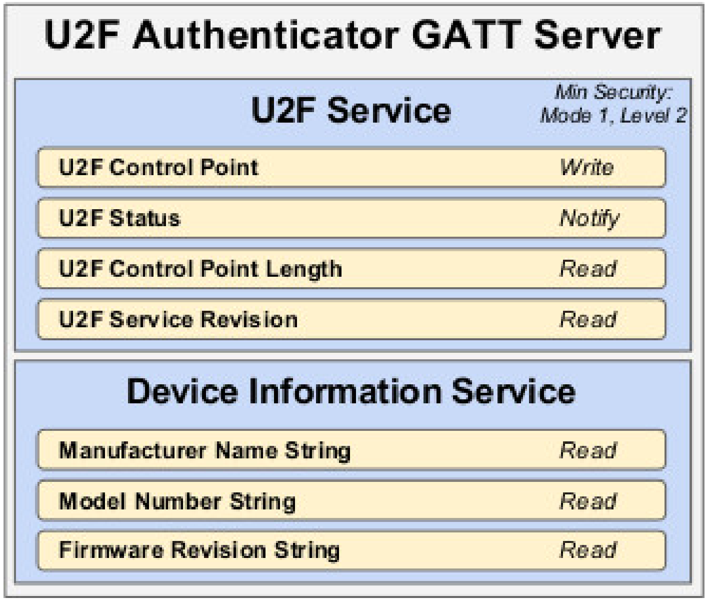
Bluetooth(tm) SIG specified UUID values SHALL be found on the Assigned Numbers website [BTASSNUM]

6.1 GATT Service Description

This profile defines two roles: FIDO Authenticator and FIDO Client.

- The FIDO Client shall be a GATT Client
- The FIDO Authenticator shall be a GATT Server

The following graphic illustrates the mandatory services and characteristics that SHALL be offered by a FIDO Authenticator as part of its GATT server:



The table below summarizes additional GATT sub-procedure requirements for a FIDO Authenticator (GATT Server) beyond those required by all GATT Servers.

GATT Sub-Procedure	Requirements
Write Characteristic Value	Mandatory
Notifications	Mandatory
Read Characteristic Descriptors	Mandatory
Write Characteristic Descriptors	Mandatory

The table below summarizes additional GATT sub-procedure requirements for a FIDO Client (GATT Client) beyond those required by all GATT Clients.

GATT Sub-Procedure	Requirements
Discover All Primary Services	(*)
Discover Primary Services by Service UUID	(*)
Discover All Characteristics of a Service	(**)
Discover Characteristics by UUID	(**)
Discover All Characteristic Descriptors	Mandatory
Read Characteristic Value	Mandatory
Write Characteristic Value	Mandatory
Notification	Mandatory
Read Characteristic Descriptors	Mandatory

Write Characteristic Descriptors	Mandatory

(*): Mandatory to support at least one of these sub-procedures.

(**): Mandatory to support at least one of these sub-procedures.

Other GATT sub-procedures may be used if supported by both client and server.

Specifics of each service are explained below. In the following descriptions: all values are big-endian coded, all strings are in UTF-8 encoding, and any characteristics not mentioned explicitly are optional.

6.1.1 U2F Service

An Authenticator SHALL implement the U2F Service described below. The UUID for the FIDO U2F GATT service is **TBD** (it is being standardized by the Bluetooth SIG). The service contains the following characteristics:

Characteristic Name	Mnemonic	Property	Length	UUID
U2F Control Point	`u2fControlPoint`	Write	Defined by Vendor (20-512 bytes)	TBD
U2F Status	`u2fStatus`	Notify	Defined by Vendor (20-512 bytes)	TBD
U2F Control Point Length	`u2fControlPointLength`	Read	2 bytes	TBD
U2F Service Revision	`u2fServiceRevision`	Read	Defined by Vendor (20-512 bytes)	0x2A28

`u2fControlPoint` is a write-only command buffer.

`u2fStatus` is a notify-only response attribute.

`u2fControlPointLength` defines the size in bytes of `u2fControlPoint`. This value SHALL be between 20 and 512.

`u2fServiceRevision` defines the revision of the U2F Service. The value is a UTF-8 string. For this version of the specification, the value `u2fServiceRevision` SHALL be `1.0` or in raw bytes: `0x312e30`.

The `u2fServiceRevision` Characteristic MAY include a Characteristic Presentation Format descriptor with format value 0x19, `UTF-8 String`.

6.1.2 Device Information Service

An Authenticator SHALL implement the Device Information Service [BTDIS] with the following characteristics:

- Manufacturer Name String
- Model Number String
- Firmware Revision String

All values for the Device Information Service are left to the vendors. However, vendors should not create uniquely identifiable values so that Authenticators do not become a method of tracking users.

6.1.3 Generic Access Service

Every Authenticator SHALL implement the Generic Access Service [BTGAS] with the following characteristics:

- Device Name
- Appearance

6.2 Protocol Overview

The general overview of the communication protocol follows:

1. Authenticator advertises the FIDO U2F service.
2. Client scans for Authenticator advertising the FIDO U2F service.
3. Client performs characteristic discovery on the Authenticator.
4. If not already paired, the Client and Authenticator SHALL perform BLE pairing and create a LTK. Authenticator SHALL only allow connections from previously bonded Clients without user intervention.
5. Client reads the `u2fControlPointLength` characteristic.
6. Client registers for notifications on the `u2fStatus` characteristic.
7. Client writes a request (*e.g.*, an enroll request) into the `u2fControlPoint` characteristic.
8. Authenticator evaluates the request and responds by sending notifications over `u2fStatus` characteristic.
9. The connection is closed by the Client or the connection times out and is closed by the Authenticator.

6.3 Authenticator Advertising Format

When advertising, the Authenticator SHALL advertise the FIDO U2F service UUID.

When advertising, the Authenticator MAY include the TxPower value in the advertisement (see [BTXPLAD]).

The advertisement MAY also carry a device name which is distinctive and user-identifiable. For example, "ACME Key" would be an appropriate name, while "XJS4" would not be.

The Authenticator SHALL also implement the Generic Access Profile [BTGAP] and Device Information Service [BTDIS], both of which also provide a user friendly name for the device which could be used by the Client.

It is not specified when or how often an Authenticator should advertise, instead that flexibility is left to manufacturers.

6.4 Requests

Clients SHOULD make requests by connecting to the Authenticator and performing a write into the `u2fControlPoint` characteristic.

6.5 Responses

Authenticators SHOULD respond to Clients by sending notifications on the `u2fStatus` characteristic.

Some Authenticators might alert users or prompt them to complete the test of user presence (*e.g.*, via sound, light, vibration, etc.) Upon receiving a valid request (containing a known key handle), the Authenticators MAY alert the user and MAY use a short delay before sending a response. In this case, before the user has completed the test of user presence, the Authenticator SHALL respond with SW_CONDITIONS_NOT_SATISFIED [U2FRAWMESSAGES]. Upon receiving an SW_CONDITIONS_NOT_SATISFIED message, the Client SHALL assume the Authenticator is still processing the command; the Client SHALL not resend the command. Until a timeout occurs, the Client SHALL NOT move on to other devices when it receives a SW_CONDITIONS_NOT_SATISFIED, as it knows this is a device that can satisfy its request.

6.6 Framing fragmentation

A single request/response sent over BLE MAY be split over multiple writes and notifications, due to the inherent limitations of BLE which is not currently meant for large messages. Frames are fragmented in the following way:

A frame is divided into an *initialization fragment* and one or more *continuation fragments*.

An initialization fragment is defined as:

Offset	Length	Mnemonic	Description
0	1	`CMD`	Command identifier
1	1	`HLEN`	High part of data length
2	1	`LLEN`	Low part of data length

3	maxLen - 3	DATA	Data

where maxLen is the maximum packet size supported by the characteristic or notification.

In other words, the start of an initialization fragment is indicated by setting the high bit in the first byte. The subsequent two bytes indicate the total length of the frame, in big-endian order. The first maxLen - 3 bytes of data follow.

Continuation fragments are defined as:

Offset	Length	Mnemonic	Description
0	1	SEQ	Packet sequence 0x00..0x7f (high bit always cleared)
1	maxLen - 1	DATA	Data

where maxLen is the maximum packet size supported by the characteristic or notification.

In other words, continuation fragments begin with a sequence number, beginning at 0, implicitly with the high bit cleared.

Example for sending a PING command with 40 bytes of data with a maxLen of 20 bytes:

Frame	Bytes
0	[810028] [17 bytes of data]
1	[00] [19 bytes of data]
2	[01] [4 bytes of data]

Example for sending a ping command with 400 bytes of data with a maxLen of 512 bytes:

Frame	Bytes
0	[810190] [400 bytes of data]

6.7 Implementation Considerations

6.7.1 Bluetooth pairing: Client considerations

As noted in the Pairing section, a disadvantage of using standard Bluetooth pairing is that the pairing is "system-wide" on most operating systems. That is, if an Authenticator is paired to a FIDO Client which resides on an operating system where Bluetooth pairing is "system-wide", then any application on that device might be able to interact with an Authenticator. This poses both security and privacy risks to users.

While Client operating system security is partly out of FIDO's scope, further revisions of this specification MAY propose mitigations for this issue.

6.7.2 Bluetooth pairing: Authenticator considerations

The method to put the Authenticator into Pairing Mode should be such that it is not easy for the user to do accidentally **especially** if the pairing method is Just Works. For example, the action could be pressing a physically recessed button or pressing multiple buttons. A visible or audible cue that the Authenticator is in Pairing Mode should be considered. As a counter example, a silent, long press of a single non-recessed button is not advised as some users naturally hold buttons down during regular operation.

6.7.3 Handling command completion

Upon completion of a command with a successful outcome, the Authenticator can assume the Client is satisfied, and MAY reset its state or power down. Upon completion with an error, the Authenticator MUST assume that the Client MAY wish to cope with the error, *e.g.,* by sending a different command or retrying the same one. If the Authenticator waits `kErrorWaitMillis` milliseconds after sending an error response, and the Client does not send an additional command, the Authenticator MAY assume the Client will not send an additional command, and reset its state and/or power down.

Constant	Value
`kErrorWaitMillis`	2000 milliseconds

6.7.4 Data throughput

BLE does not have particularly high throughput, this can cause noticeable latency to the user if request/responses are large. Some ways that implementers can reduce latency are:

- Support the maximum MTU size allowable by hardware (up to the 512 bytes max from the BLE specifications).
- Make the attestation certificate as small as possible, do not include unnecessary extensions.

6.7.5 Advertising

Though the standard doesn't appear to mandate it (in any way that we've found thus far), advertising and device discovery seems to work better when the Authenticators advertise on all 3 advertising channels and not just one.

7. Bibliography

[BTASSNUM] Bluetooth Assigned Numbers. https://www.bluetooth.org/en-us/specification/assigned-numbers

[BTCORE] Bluetooth Core Specification 4.1. see https://www.bluetooth.org/en-us/specification/adopted-specifications

[BTDIS] Device Information Service V1.1. see https://www.bluetooth.org/en-us/specification/adopted-specifications

[BTGAP] Generic Access Profile. Bluetooth Core Specification 4.1, Volume 3, Part C, Section 12. see https://www.bluetooth.org/en-us/specification/adopted-specifications

[BTGAS] Generic Access Service. Bluetooth Core Specification 4.1, Volume 3, Part C, Section 12. see https://developer.bluetooth.org/gatt/services/Pages/ServiceViewer.aspx?u=org.bluetooth.service.generic_access.xml

[BTXPLAD] Bluetooth TX Power AD Type. Bluetooth Core Specification 4.1, Volume 3, Part C, Section 11. see https://www.bluetooth.org/en-us/specification/adopted-specifications

[U2FRAWMESSAGES] Dirk Balfanz, Jakob Ehrensvard. FIDO U2F Raw Message Formats, Aug 2014.

A. References

A.1 Normative references

[ECMA-262]
ECMAScript Language Specification, Edition 5.1. June 2011. URL: http://www.ecma-international.org/publications/standards/Ecma-262.htm

[FIDOGlossary]
R. Lindemann, D. Baghdasaryan, B. Hill, J. Hodges, *FIDO Technical Glossary*. FIDO Alliance Proposed Standard. URLs:
HTML: fido-glossary.html
PDF: fido-glossary.pdf

[RFC2119]
S. Bradner. *Key words for use in RFCs to Indicate Requirement Levels*. March 1997. Best Current Practice. URL: https://tools.ietf.org/html/rfc2119

[WebIDL]
Cameron McCormack. *Web IDL*. 19 April 2012. W3C Candidate Recommendation. URL: http://www.w3.org/TR/WebIDL/