FIDO U2F HID Protocol Specification

FIDO Alliance Proposed Standard 09 October 2014

This version:
http://www.fidoalliance.org/specs/fido-u2f-hid-protocol-v1.0-ps-20141009.html
Previous version:
http://www.fidoalliance.org/specs/fido-u2f-hid-protocol-v1.0-ID-20141009.html
Editors:
Jakob Ehrensvärd, Yubico
John Kemp, FIDO Alliance

The English version of this specification is the only normative version. Non-normative translations may also be available.

Abstract

U2FHID protocol description and implementation specification

The purpose of this documentation is to provide a complete specification how to implement the U2FHID protocol, where FIDO U2F messages are framed for USB transport, using the HID protocol. General FIDO and U2F- concepts, semantics, meaning is beyond the scope of this document and for information on these topics, please refer to the appropriate related documentation.

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 Proposed Standard. If you wish to make comments regarding this document, please Contact Us. All comments are welcome.

Implementation of certain elements of this Specification may require licenses under third party intellectual property rights, including without limitation, patent rights. The FIDO Alliance, Inc. and its Members and any other contributors to the Specification are not, and shall not be held, responsible in any manner for identifying or failing to identify any or all such third party intellectual property rights.

THIS FIDO ALLIANCE SPECIFICATION IS PROVIDED “AS IS” AND WITHOUT ANY WARRANTY OF ANY KIND, INCLUDING, WITHOUT LIMITATION, ANY EXPRESS OR IMPLIED WARRANTY OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

This document has been reviewed by FIDO Aliance Members and is endorsed as a Proposed Standard. It is a stable document and may be used as reference material or cited from another document. FIDO Alliance's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment.

Table of Contents

1. Document Information

1.1 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].

Symbolic constants such as **U2FHID_MSG** which are referred to when defining messages in this documents have their values defined in [U2FHIDHeader] in the bibliography.

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

1.1.1 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].

1.2 Definitions

Term	Definition
U2F	Universal Second Factor
USB	Universal Serial Bus
HID	Human Interface Device. A specification of typical USB devices used for human interaction, such as keyboards, mice, joysticks etc.
U2FHID	U2F transport over HID as defined by this document

2. U2FHID protocol implementation

This description does not describe the actual raw U2F messages, semantics and functionality but rather how such messages are framed for HID transport. The raw U2F messages are defined in [U2FRawMsgs] in the bibliography.

2.1 U2FHID implementation rationale

The U2FHID protocol is designed with the following design objectives in mind

- Driver-less installation on all major host platforms
- Multi-application support with concurrent application access without the need for serialization and centralized dispatching.
- Fixed latency response and low protocol overhead
- Scalable method for U2FHID device discovery

Since HID data is sent as interrupt packets and multiple applications may access the HID stack at once, a non-trivial level of complexity has to be added to handle this.

2.2 Protocol structure and data framing

The U2F protocol is designed to be concurrent and state-less in such a way that each performed function is not dependent on previous actions. However, there has to be some form of "atomicity" that varies between the characteristics of the underlying transport protocol, which for the U2FHID protocol introduces the following terminology:

- Transaction
- Message
- Packet

A **transaction** is the highest level of aggregated functionality, which in turn consists of a request, followed by a response message. Once a request has been initiated, the transaction has to be entirely completed before a second transaction can take place and a response is never sent without a previous request.

Request- and response **messages** are in turn divided into individual fragments,

known as **packets**. The packet is the smallest form of protocol data unit, which in the case of U2FHID are mapped into HID reports.

2.3 Concurrency and channels

Additional logic and overhead is required to allow a U2FHID device to deal with multiple "clients", i.e. multiple applications accessing the single resource through the HID stack. Each client communicates with a U2FHID device through a logical **channel**, where each application uses a unique 32-bit **channel identifier** for routing- and arbitration purposes.

A channel identifier is allocated by the U2F device to ensure its system-wide uniqueness. The actual algorithm for generation of channel identifiers is vendor specific and not defined by this specification.

Channel ID 0 is reserved and `0xffffffff` is reserved for broadcast commands, i.e. at the time of channel allocation.

2.4 Message- and packet structure

Packets are one of two types, **initialization packets** and **continuation packets**. As the name suggests, the first packet sent in a message is an initialization packet, which also becomes the start of a transaction. If the entire message does not fit into one packet (including the U2FHID protocol overhead), one or more continuation packets have to be sent in strict ascending order to complete the message transfer.

A message sent from a host to a device is known as a **request** and a message sent from a device back to the host is known as a **response**. A request always triggers a response and response messages are never sent ad-hoc, i.e. without a prior request message.

The request and response messages have an identical structure. A transaction is started with the initialization packet of the request message and ends with the last packet of the response message.

Packets are always fixed size (defined by the endpoint- and HID report descriptors) and although all bytes may not be needed in a particular packet, the full size always has to be sent. Unused bytes should be set to zero.

An initialization packet is defined as

Offset	Length	Mnemonic	Description
0	4	CID	Channel identifier
4	1	CMD	Command identifier (bit 7 always set)
5	1	BCNTH	High part of payload length
6	1	BCNTL	Low part of payload length
7	(s - 7)	DATA	Payload data (s is equal to the fixed packet size)

The command byte has always the highest bit set to distinguish it from a continuation packet, which is described below.

A continuation packet is defined as

Offset	Length	Mnemonic	Description
0	4	CID	Channel identifier
4	1	SEQ	Packet sequence 0x00..0x7f (bit 7 always cleared)
5	(s - 5)	DATA	Payload data (s is equal to the fixed packet size)

With this approach, a message with a payload less or equal to (s - 7) may be sent as one packet. A larger message is then divided into one or more continuation packets, starting with sequence number 0, which then increments by one to a maximum of 127.

With a packet size of 64 bytes (max for full-speed devices), this means that the maximum message payload length is 64 - 7 + 128 * (64 - 5) = 7609 bytes.

2.5 Arbitration

In order to handle multiple channels and clients concurrency, the U2FHID protocol has to maintain certain internal states, block conflicting requests and maintain protocol integrity. The protocol relies on each client application (channel) behaves politely, i.e. does not actively act to destroy for other channels. With this said, a malign- or malfunctioning application can cause issues for other channels. Expected errors and potentially stalling applications should however be handled properly.

2.5.1 Transaction atomicity, idle- and busy states.

A transaction always consists of three stages:

1. A message is sent from the host to the device
2. The device processes the message
3. A response is sent back from the device to the host

The protocol is built on the assumption that a plurality of concurrent applications may try ad-hoc to perform transactions at any time, with each transaction being atomic, i.e. it cannot be interrupted by another application once started.

The application channel that manages to get through the first initialization packet when the device is in idle state will keep the device locked for other channels until the last packet of the response message has been received. The device then returns to idle state, ready to perform another transaction for the same or a different channel. Between two transactions, no state is maintained in the device and a host application must assume that any other process may execute other transactions at any time.

If an application tries to access the device from a different channel while the device is busy with a transaction, that request will immediately fail with a busy-error message sent to the requesting channel.

2.5.2 Transaction timeout

A transaction has to be completed within a specified period of time to prevent a stalling application to cause the device to be completely locked out for access by other applications. If for example an application sends an initialization packet that signals that continuation packets will follow and that application crashes, the device will back out that pending channel request and return to an idle state.

2.5.3 Transaction abort and re-synchronization

If an application for any reason "gets lost", gets an unexpected response or error, it may at any time issue an abort-and-resynchronize command. If the device detects a SYNC command during a transaction that has the same channel id as the active transaction, the transaction is aborted (if possible) and all buffered data flushed (if any). The device then returns to idle state to become ready for a new transaction.

2.5.4 Packet sequencing

The device keeps track of packets arriving in correct and ascending order and that no expected packets are missing. The device will continue to assemble a message until all parts of it has been received or that the transaction times out. Spurious continuation packets appearing without a prior initialization packet will be ignored.

2.6 Channel locking

In order to deal with aggregated transactions that may not be interrupted, such as vendor specific tunneling of APDUs, a channel lock command may be implemented. By sending a channel lock command, the device prevents other channels from communicating with the device until the channel lock has timed out or been explicitly unlocked by the application.

This feature is optional and has not to be considered by general U2F HID applications.

2.7 Protocol version and compatibility

The U2FHID protocol is designed to be extensible, yet maintaining backwards compatibility to the extent it is applicable. This means that a U2FHID host shall support any version of a device with the command set available in that particular version.

3. HID device implementation

This description assumes knowledge of the USB- and HID specifications and is

intended to provide the basics for implementing a U2FHID device. There are several ways to implement USB devices and reviewing these different methods is beyond the scope of this document. This specification targets the interface part, where a device is regarded as either a single- or multiple interface (composite) device.

The description further assumes (but is not limited to) a full-speed USB device (12 Mbit/s). Although not excluded per se, USB low-speed devices are not practical to use given the 8-byte report size limitation together with the protocol overhead.

3.1 Interface- and endpoint descriptors

The device implements two endpoints (except the control endpoint 0), one for IN- and one for OUT transfers. The packet size is vendor defined, but the reference implementation assumes a full-speed device with two 64-bytes endpoints.

Interface Descriptor

Mnemonic	Value	Description
bNumEndpoints	2	One IN- and one OUT endpoint
bInterfaceClass	0x03	HID
bInterfaceSubClass	0x00	No interface subclass
bInterfaceProtocol	0x00	No interface protocol

Endpoint 1 descriptor

Mnemonic	Value	Description
bmAttributes	0x03	Interrupt transfer
bEndpointAdresss	0x01	1, OUT
bMaxPacketSize	64	64 bytes packets
bInterval	5	Poll every 5 millisecond

Endpoint 2 descriptor

Mnemonic	Value	Description
bmAttributes	0x03	Interrupt transfer
bEndpointAdresss	0x81	1, IN
bMaxPacketSize	64	64 bytes packets
bInterval	5	Poll every 5 millisecond

The actual endpoint order, intervals, endpoint numbers and endpoint packet size may be defined freely by the vendor and the host application is responsible for querying

these values and handle these accordingly. For the sake of clarity, the values listed above are used in the following examples.

3.2 HID report descriptor and device discovery

A HID report descriptor is required for all HID devices, even though the reports and their interpretation (scope, range, etc.) makes very little sense from an operating system perspective. The U2FHID just provides two "raw" reports, which basically map directly to the IN and OUT endpoints. However, the HID report descriptor has an important purpose in U2FHID, as it is used for device discovery.

For the sake of clarity, a bit of high-level C-style abstraction is provided

> EXAMPLE 1
>
> ```
> // HID report descriptor
>
> const uint8_t HID_ReportDescriptor[] = {
>   HID_UsagePage ( FIDO_USAGE_PAGE ),
>   HID_Usage ( FIDO_USAGE_U2FHID ),
>   HID_Collection ( HID_Application ),
>   HID_Usage ( FIDO_USAGE_DATA_IN ),
>   HID_LogicalMin ( 0 ),
>   HID_LogicalMaxS ( 0xff ),
>   HID_ReportSize ( 8 ),
>   HID_ReportCount ( HID_INPUT_REPORT_BYTES ),
>   HID_Input ( HID_Data | HID_Absolute | HID_Variable ),
>   HID_Usage ( FIDO_USAGE_DATA_OUT ),
>   HID_LogicalMin ( 0 ),
>   HID_LogicalMaxS ( 0xff ),
>   HID_ReportSize ( 8 ),
>   HID_ReportCount ( HID_OUTPUT_REPORT_BYTES ),
>   HID_Output ( HID_Data | HID_Absolute | HID_Variable ),
> HID_EndCollection
> };
> ```

A unique **Usage Page** is defined for the FIDO alliance and under this realm, a U2FHID **Usage** is defined as well. During U2FHID device discovery, all HID devices present in the system are examined and devices that match this usage pages and usage are then considered to be U2FHID devices.

The length values specified by the `HID_INPUT_REPORT_BYTES` and the `HID_OUTPUT_REPORT_BYTES` should typically match the respective endpoint sizes defined in the endpoint descriptors.

4. U2FHID commands

The U2FHID protocol implements the following commands. These commands are not related to U2F commands, which are send using the U2FHID_MSG command

4.1 Mandatory commands

The following list describes the minimum set of commands required by an U2FHID device. Optional- and vendor-specific commands may be implemented as described in respective sections of this document.

4.1.1 U2FHID_MSG

This command sends an encapsulated U2F message to the device. The semantics of the data message is defined in the U2F protocol specification.

Request

CMD	U2FHID_MSG
BCNT	4..n
DATA	n bytes

Response at success

CMD	U2FHID_MSG
BCNT	2..n
DATA	N bytes

4.1.2 U2FHID_INIT

This command synchronizes a channel and optionally requests the device to allocate a unique 32-bit channel identifier (CID) that can be used by the requesting application during its lifetime. The requesting application generates a nonce that is used to match the response. When the response is received, the application compares the sent nonce with the received one. After a positive match, the application stores the received channel id and uses that for subsequent transactions.

To allocate a new channel, the requesting application shall use the broadcast channel U2FHID_BROADCAST_CID. The device then responds the newly allocated channel in the response, using the broadcast channel.

Request

CMD	U2FHID _INIT
BCNT	8
DATA	8 byte nonce

Response at success

CMD	U2FHID _INIT

BCNT	17 (see note below)
DATA	8 byte nonce
DATA+8	4 byte channel ID
DATA+12	U2FHID protocol version identifier
DATA+13	Major device version number
DATA+14	Minor device version number
DATA+15	Build device version number
DATA+16	Capabilities flags

The protocol version identifies the protocol version implemented by the device. An U2FHID host shall accept a response size that is longer than the anticipated size to allow for future extensions of the protocol, yet maintaining backwards compatibility. Future versions will maintain the response structure to this current version, but additional fields may be added.

The meaning and interpretation of the version number is vendor defined.

The following device capabilities flags are defined. Unused values are reserved for future use and must be set to zero by device vendors.

CAPABILITY_WINK	Implements the WINK function

4.1.3 U2FHID_PING

Sends a transaction to the device, which immediately echoes the same data back. This command is defined to be an uniform function for debugging-, latency- and performance measurements.

Request

CMD	U2FHID_PING
BCNT	0..n
DATA	n bytes

Response at success

CMD	U2FHID_PING
BCNT	n
DATA	N bytes

4.1.4 U2FHID_ERROR

This command code is used in response messages only.

CMD	U2FHID_ERROR
BCNT	1
DATA	Error code

The following error codes are defined

ERR_INVALID_CMD	The command in the request is invalid
ERR_INVALID_PAR	The parameter(s) in the request is invalid
ERR_INVALID_LEN	The length field (BCNT) is invalid for the request
ERR_INVALID_SEQ	The sequence does not match expected value
ERR_MSG_TIMEOUT	The message has timed out
ERR_CHANNEL_BUSY	The device is busy for the requesting channel

4.2 Optional commands

The following commands are defined by this specification but are optional and does not have to be implemented.

4.2.1 U2FHID_WINK

The wink command performs a vendor-defined action that provides some visual- or audible identification a particular U2F device. A typical implementation will do a short burst of flashes with a LED or something similar. This is useful when more than one device is attached to a computer and there is confusion which device is paired with which connection.

Request

CMD	U2FHID_WINK
BCNT	0
DATA	N/A

Response at success

CMD	U2FHID_WINK
BCNT	0
DATA	N/A

4.2.2 U2FHID_LOCK

The lock command places an exclusive lock for one channel to communicate with the device. As long as the lock is active, any other channel trying to send a message will fail. In order to prevent a stalling- or crashing application to lock the device indefinitely, a lock time up to 10 seconds may be set. An application requiring a longer lock has to send repeating lock commands to maintain the lock.

Request

CMD	U2FHID_LOCK
BCNT	1
DATA	Lock time in seconds 0..10. A value of 0 immediately releases the lock

Response at success

CMD	U2FHID_LOCK
BCNT	0
DATA	N/A

4.3 Vendor specific commands

A U2FHID may implement additional vendor specific commands that are not defined in this specification, yet being U2FHID compliant. Such commands, if implemented must have a command in the range between U2FHID_VENDOR_FIRST and U2FHID_VENDOR_LAST.

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 *FIDO Technical Glossary v1.0*. FIDO Alliance Implementation Draft. URL: http://fidoalliance.org/specs/fido-glossary-v1.0-ps-20141009.pdf

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

[U2FHIDHeader]
J. Ehrensvard, *FIDO U2F HID Header Files v1.0*. FIDO Alliance Implementation Draft. URL: https://fidoalliance.org/specs/fido-u2f-u2f_hid.h-v1.0-ps-20141009.pdf

[U2FRawMsgs]
D. Balfanz, J. Ehrensvard, *FIDO U2F Raw Message Formats v1.0*. FIDO Alliance Implementation Draft. URL: http://fidoalliance.org/specs/fido-u2f-raw-message-formats-v1.0-ps-20141009.pdf

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