Path Verification for FDO (PV4FDO)

Review Draft, January 22, 2026

This version:
https://fidoalliance.org/specs/FDO/pv4fdo-v1.0-rd-20260122/pv4fdo-v1.0-rd-20260122.html

Issue Tracking:
GitHub

Editors:
Gerardo Diaz Cuellar (Microsoft)
Geoffrey Cooper (Intel)
Brad Goodman (Dell)

Abstract

Path Verication for FDO (PV4FDO) is a protocol to obtain evidence of the path through the network that the FDO protocol takes. It is intended to increase confidence in an attestation from a given device, by giving evidence that the device is part of the intended network topology.

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://fidoalliance.org/specifications/.

This document was published by the FIDO Alliance as a Review Draft Specification. 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.

This is a Review Draft Specification and is not intended to be a basis for any implementations as the Specification may change. Permission is hereby granted to use the Specification solely for the purpose of reviewing the Specification. No rights are granted to prepare derivative works of this Specification. Entities seeking permission to reproduce portions of this Specification for other uses must contact the FIDO Alliance to determine whether an appropriate license for such use is available.

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.

Table of Contents

1. Introduction # {pvIntro}§

Path Verification for FDO (PV4FDO) is a step on the way to addressing the "proof of possession" problem in FDO: whether a given FDO device being onboarded corresponds to a device that is authorized to be physically in its installation position.

Establishing proof of possession for a device in hand is easy; when an automated protocol establishes a link to an external Device, it becomes harder. The Internet protocols provide wide connectivity and it is possible that the selected Device, running FDO, is not in the infrastructure where it is supposed to be.

PV4FDO obtains specific evidence on the path taken for a given FDO transaction. Used properly, it can increase the confidence that the FDO Device is in the intended place in the target infrastructure, and so help to address the "proof of possession" problem.

The protocol works by allowing the FDO server’s infrastructure to determine whether a given TCP-based connection (e.g., the FDO connection) passed through a particular waypoint in the network infrastructure. This test can be performed at the machine acting as the FDO server or at another machine that is assisting FDO server(s) in this way.

No provision is provided for redundant IP protocol paths, so this protocol is most useful when applied from a network choke point through which FDO must pass. When redudant paths exist, the stability of IP routing can still make it feasible to use this protocol with careful analysis of the results.

2. Other Internet Path Verification Mechanisms # {pvPathVerif}§

This protocool performs a limited (one point!) path verification. Path verification is a larger field of development in the Internet. This protocol does not intend to address the larger issue. Some examples of other path verification mechanisms:

- The BGP protocol addresses path verification at the BGP boundary
- IP source routing provides a more complete path verification mechanism
- Proof-of-transit protocols with pre-provisioned secrets can provide step-by-step path verfication

This protocol is most closely related existing Internet proof-of-transit, but operates at only a single waypoint.

3. Vocabulary # {pvVocabulary}§

Device: the device being onboarded with the FDO protocol,usually assumed to be an headless device. The device often hosts a protocol entity, also called Device. By convention in this document, we capitalize the protocol entity.

FDO Target / Connection Target: The target FDO server whose traffic is being monitored at the PV4FDO Monitoring Point.

Installer (person): a person who is charged with physical installation of network devices.

PVGuid: A unique ID that identifies a Monitoring Point for a given connection.

Mirror Port: A port on a switch that is programmed to receive a mirror of some specified traffic at other ports on the switch.

Monitoring Point: The point in the target network where traffic is being monitored.

Monitoring Extent: How far into each target connection at the Monitoring Point is being examined.

Path Verification: the determination of whether a given network connection passes through the intended path on the Internet.

PV4FDO port: a given TCP port number used for the PV4FDO service on a given target machine. Port 61 may be a useful and available (site-specific) port for PV4FDO.

SPAN port: a switch mirror port. SPAN stands for "Switched Port Analyzer."

TLS Record Protocol: the layered protocol on which TLS is created. For information, record protocol, reference TLS specifications, such as RFC8446, section 5.

4. Threat Mitigation # {pvThreatMitg}§

This section indicates threats to FDO onboarding that can be mitigated using path verification, and specifically using this protocol.

In the simplest situation an incorrect path is a mistake rather than an attack, although the mistake might be discovered and used by attackers (and so become a threat). An incorrect path results when a target Device is intended to be attached to a given network attachment point, but is mistakenly connected to another place in the network. For example, an Installer (person) might attach the device to the wrong cable by mistake. If the network routing is well connected, the device can still onboard using FDO and appear completely healthy when accessed over the network. However, any network-specific function of the device would be different from the intent of the Installer. For example, the device might be in a different security realm and be subject to unanticipated attacks. The incorrect place on the network would likely be visible from the Device' IP address, but this information might not be noticed by Installers or other network maintenance personnel.

In a more malicious example, a "doorstep attack", an attacker swaps the attacker's box for the victim's box, e.g., when the victim is not paying attention. A box containing an expected new Device is delivered to the expected physical location. Before the box is noticed by the recipient, it is substituted for another box. The device delivered (e.g., left on the doorstep) is believed to be the expected delivery, and is installed on the local network. Now the attacker's device is installed in the victim's network. Meanwhile the victim's device can be installed in the attacker's network, so the victim can actually see and authenticate the intended device on the Internet. Since FDO has no way to know about the intended device' mislocation on the Internet, the victim's device runs FDO and FDO unwittingly provisions a device in the attacker's network.

The PV4FDO protocol can reveal these situations by verifying that a given connection passed through a known "choke point" on an intended segment of the Internet. For example, this point might be inside a particular company branch building, verifying that the deployed device, performing FDO, is physically inside this building.

5. Overall Approach # {pvOverall}§

In this protocol, the network owner establishes at least one waypoint where the PV4FDO protocol is deployed. The waypoint can be (or advertise itself as) an IP router, forcing traffic to go through it. The network infrastructure can also force a copy (mirror) of the traffic to go through the waypoint, such as using a switch's SPAN (Switched Port ANalayzer) port.

PV4FDO nodes are configured with:

- A server port for PV4FDO.
- An access token mechanism for nodes to use with HTTP POST.
- A monitoring location ("Monitoring Point") in the infrastructure where the PV4FDO node is attached. This location must represent a chokepoint in the network, as described above.
- A means of monitoring the FDO traffic over HTTP and TLS. For example, PV4FDO can be integrated into an Internet router, or might use switch-based routing to force traffic to go through the monitoring point. A switch based mirror port (aka SPAN port) can also be used.
- A configured limit of how far into each monitored connection to examine ("Monitoring Extent")

When a new connection is sensed by PV4FDO, it opens a data structure and starts computing the hash of the FDO or TLS data in the connection. The data structure is indexed by a connection ID (see below).

*** The hash is computed on traffic "upbound", from the client to the server.

As an implementation-specific extension, it is possible for a PV4FDO implementation to continue monitoring a connection *after* the monitoring extent, generating additional PV4FDO messages for the same FDO connection, using the same PVGuid. How these additional PV4FDO hashes are reconciled is implementation-specific.

A PVGuid is computed by hashing the FDO/TLS data being monitored for the FDO TO2 protocol. For a FDO session encapsulated within a TLS connection, the first TLS record (the ClientHello) contains the Client Random, which contains sufficient entropy. So the PVGuid is computed based only on the first the TLS record containing the ClientHello.

Note that it is only necessary to parse the TLS record layer to create this hash. In all cases, when hashing TLS data the data MUST be hashed based on **TLS record boundaries**.

For an FDO session that is not encapsulated inside of TLS, the connection data MUST be hashed until:

- for TO1, the TO1.ProveToRV message is hashed
- for TO2, the TO2.HelloDevice message is transmitted

The TO0 protocol does not need to be protected by PV4FDO.

In both cases, it is only necessary to hash the client to server data in the TCP connection.

> NOTE: Since we are only looking for entropy, there is no need to interpret or (certainly not (!)) to decrypt the TLS session. We only need to hash the raw data, whether encrypted (FDO over TLS) or not (FDO directly on TCP).

In addition to computing the PVGuid, the PV4FDO implementation computes a separate hash of the TCP-level client to server data in the connection, up until the chosen (configured) monitoring extent.

It may be possible to use a single hash computation that determines the PVGuid then continues to hash to the monitoring extent.

PV4FDO sends this hash to the FDO target (i.e., the target of the original monitored FDO or TLS/FDO traffic) using the configured PV4FDO port. The receiving node has also received the same FDO or TLS data, and can compute its own hash, based on its received data. By comparing the hashes, the receiving node can verify that the data in the connection transited the configured monitoring point.

If the received hash of monitored traffic does not match the computed hash in the FDO target, the FDO target SHOULD drop the FDO connection, forcing the sending to retry.

If the FDO target does not implement PV4FDO, the connection containing the PV4FDO hash data is dropped and no PV4FDO benefit is received.

If the FDO target performs a network transaction with a network that is known to be covered by PV4FDO but does `not` receive a PV4FDO message, the connection is now suspected of coming from outside the expected infrastructure. In this case, the receiver MUST log an error message and SHOULD reject the high-level action requested by the protocol.

The following information is outside the scope of this protocol:

- How to ensure that the receiving nodes implement PV4FDO
- Whether the proxy can suppress sending hash data to nodes that do not implement PV4FDO

The choice of hash is as follows:

```
Hash        Hash-ID
--------------------
SHA256       -16
SHA384       -43
```

Other hash types can be added as needed.

6. Protocol Technical Description # {pvTechDescr}§

PV4FDO_Proxy -> TO1/TO2 Service URL: /fdo/communication/client Request Method: HTTP Post Message Format: Proxy.SendHash

The HTTP POST body contains the following structure. The structure can be expressed in CBOR or JSON (CBOR can be more natural on the client, since FDO itself uses CBOR).

```
Proxy.SendHash = [
pvguid,
hash,
uint hashedBytes ;; number of bytes comprised in the hash
]
```

Http Headers:

- Authorization: Bearer [Access_Token]
- This is mandatory. The access token is used for authentication.
- Content-Type: application/cbor or application/json depending which data format is used

The form, use and configuration of the access token is a site-based parameter that is outside the scope of this specification.

7. Message Meaning:§

In TLS FDO sessions:

P4FDO proxy sends the hash value of the TLS data captured from the FDO Agent, at TLS record boundaries. The API is called whenever the FDO Agent sends out new data. The P4FDO Proxy retrieves the host name and port of the TO1/TO2 Service when it sets up the connection between the FDO Agent and the TO1/TO2 Service.

The Proxy will retry if the status code of the response is not 200. The number of times the proxy will retry, and the delay waiting for the 200, is implementation specific.

The pvguid is the hash of the TLS record containing Client Hello Random, e.g. 48A04FF76FBE35F664B27DF9AF1E045662A7CAC730021EAAF388BAE1B2133C1F.

The hash is the hash value of the accumulated TLS data captured from the FDO Agent, e.g. 6ABF8C41DE4AE49AE4ADEC53061BF053AAA515125D35A51DD3B149FFFBB41184

> NOTE: Implementors might decide to prefer FDO sessions over TLS for defense in depth.

TO1/TO2 Service -> Proxy

The TO1/TO2 Service responds with status code 200 if it accepts the post request; Otherwise, it responds with 400 (e.g. when access token is invalid or when there are too many requests). There is no mandatory headers. The response body is empty.

Example 1

```
Request URL: "https://mockto1service.com:57900/fdo/communication/client"
Request Method: POST
Status Code: 200 OK

Request Headers:
Authorization: "Bearer good_access_token"
Content-Length: 143
Content-Type: "application/cbor"
Host: "mockto1service.com:57900"

Request Body:
A2646775696478404338434231314130363734383433443139394543413444304244453643383233\\
44413745314144393344433545374244373737444436443434393543314443316468617368784039\\
44343745374539323436413946393130364431374434414544323832433137463044423436434343\\
3431354341413333454143353044334536333936413944

{"pvguid": "C8CB11A0674843D199ECA4D0BDE6C823DA7E1AD93DC5E7BD777DD6D4495C1DC1",
"hash": "9D47E7E9246A9F9106D17D4AED282C17F0DB46CCC415CAA33EAC50D3E6396A9D",
"hashedBytes": 256 }
```

> NOTE: This request body is the byte string of the CBOR data. It is a map that contains 3 key-value pairs: a pvguid (C8CB1...), a hash (9D47E...) and the number of bytes covered by the hash (hashBytes, here 256).

Example 2

```
Request URL: "https://mockto1service.com:57900/fdo/communication/client"
Request Method: POST
Status Code: 400 Bad Request

Request Headers:
Authorization: "Bearer bad_access_token"
Content-Length: 143
Content-Type: "application/cbor"
Host: "mockto1service.com:57900"

Request Body:
A2646775696478404338434231314130363734383433443139394543413444304244453643383233\\
44413745314144393344433545374244373737444436443434393543314443316468617368784037\\
38364343354342373631394636353337313045333930333945363043433930313945303145424344\\
4646423044373244394645303146453436333041333941

{"pvguid": "C8CB11A0674843D199ECA4D0BDE6C823DA7E1AD93DC5E7BD777DD6D4495C1DC1",
"hash": "786CC5CB7619F653710E39039E60CC9019E01EBCDFFB0D72D9FE01FE4630A39A",
"hashedBytes": 258 }
```

> NOTE: this request body is the byte string of the cbor data. It is a map that contains 3 pairs of key-value: a pvguid (C8CB1...), a hash (786CC...), and the number of bytes covered by the hash (hashBytes, here 258).