IEEE P802.11
Wireless LANs

|  |
| --- |
| Post-Quantum Device Provisioning Protocol |
| Date: 2025-05-30 |
| Author(s): |
| Name | Affiliation | Address | Phone | email |
| Dan Harkins | HPE |  |  |  |
|  |  |  |  |  |

Abstract

This submission describes what a PQC version of the Device Provisioning Protocol’s (DPP) Authentication protocol might look like.

The Device Provisioning Protocol (DPP) is a secure on-boarding protocol aimed primarily, though not exclusively, at IoT. It allows for zero touch provisioning of devices with and without a functional user interface in a robust and misuse resistant manner. DPP is defined in a Technical Specification from the Wi-Fi Alliance [1].

DPP has 4 distinct phases: discovery, authentication, configuration, network access. The DPP Authentication protocol uses elliptic curve cryptography to achieve authentication and derive a secret used to protect the subsequent configuration step. A post quantum cryptography (PQC) version of the DPP Authentication protocol is described here.

Notation:

xsky, XPKY is a PQC keypair

x = s/S for static or e/E for ephemeral

y = i/I for initiator or r/R for responder

AES-SIV-enc() and AES-SIV-dec() are the nonce-less modes of AES-SIV [3] using a 512-bit key.

H() and HKDF-(extract/expand) use a hash algorithm that depends on the security level of the PQC parameter set being used. For example,

|  |  |
| --- | --- |
|  PQC Parameter Set |  Hash Algorithm |
| ML-KEM-512 | SHA256 |
| ML-KEM-768 | SHA384 |
| ML-KEM-1024 | SHA512 |
| mceliece348864  | SHA256 |
| mceliece460896  | SHA384 |
| mceliece6688128 | SHA512 |

Due to the massive size of its public keys and the extremely small size of its ciphertexts, Classic McEliece is uniquely suitable for the static keys used in PQC DPP but is ill-suited to perform the ephemeral portion of the exchange. Otherwise, the PQC DPP Authentication protocol should be able to complete without fragmentation and reassembly of component frames.

The peers each have a static public key which uniquely identifies it through possession of the corresponding secret key. It is the ability of an entity to demonstrate knowledge of the secret key (through a successful KEM.Decaps() invocation) that provides authentication without digital signatures. It is computationally infeasible for another entity to impersonate either peer, knowledge of an entity’s public key is not sufficient.

This protocol assumes the Configurator has obtained the Enrollee’s trusted static PQC public key in a “bootstrapping protocol” as described in [1], optionally the Enrollee can obtain and trust the Configurator’s PQC public key. If the Enrollee has obtained the Configurator’s public key it can achieve strong cryptographic authentication whereas if has not, it achieves a weaker form of authentication than the Configurator gets under the Resurrecting Duckling security model [2].

The Enrollee has performed sskr, SPKr = KEM.KeyGen and the Configurator has performed sski, SPKi = KEM.KeyGen(). The Configurator has bootstrapped SPKr and the Enrollee may optionally bootstrap SPKi if it desires to do mutual authentication (see [1]). The Configurator takes the DPP role of “initiator” while the Enrollee takes the DPP role of “responder” for the PQC DPP Authentication protocol.

When the DPP service starts (e.g. at bootup when a device lacks a suitable credential to get on a network) an Enrollee scans for SSIDs which also advertise the DPP Configurator Connectivity element in beacons and probe responses. It builds a “chirp” list of the union of all the bands on which such an AP exists and a single default channel in each band it supports. It then sends a “chirp” on each channel in its chirp list and dwells for a short period of time to see if there is a response.

A Configurator that receives a chirp and is able to recognize it (by constructing all instances of SHA256(“chirp” | pk) where pk is the public key for each device it has bootstrapped) initiates the PQC DPP Authentication protocol.

The exchange is identified by a concatenation of the two session ids, sidi and sidr which need not be longer than 16 octets.

**Security Goals of PQC DPP**

The PQC DPP Authentication protocol aims to achieve the following:

Authentication: the Configurator will have the ability to uniquely verify the identity of the Enrollee—that is, it is computationally infeasible that a 3rd party could impersonate the Enrollee to the Configurator. If the Enrollee has bootstrapped the Configurator’s public key, the Enrollee will have the ability to uniquely verify the identity of the Configurator. If it chooses to engage in the PQC DPP Authenticaiton exchange without bootstrapping the Configurator’s public key it will have a weaker assurance that the Configurator is a trusted entity given that it knows the Enrollee’s (somewhat secret) public key.

Consistency: Successful completion of the PQC DPP Authentication exchange will result in the two parties establishing a secret session key and a consistent view of the identity of each peer. A successful run of the protocol will result in party A having a session with B and B having a matching session with A.

Session-Key Secrecy: An adversary will not be able to learn anything about a session key. A session key will be indistinguishable from random to an attacker that interferes with or observes a protocol run.

Identity Protection: The long-term identities of the peers in the form of their static public keys are never exposed or passed in the clear. A limited amount of traffic analysis is possible due to the fact that hashes of the long term identities are passed in the clear. This is mitigated by the fact that DPP is a “one-off” protocol and once a credential for network access is obtained it is not run again.

When operating under the Resurrecting Duckling security model the PQC DPP Authentication exchange is not resistant to key-compromise impersonation (KCI) attacks since the Enrollee’s assurance is simply that the Configurator knows its (semi-secret) public key and therefore impersonation of any Configurator to the Enrollee is possible. When the Enrollee performs mutual authentication the PQC DPP Authentication exchange is resistant to KCI attacks.

The PQC DPP Authentication protocol is resistant to unknown key share (UKS) attacks, also known as identity mis-binding attacks against the Configurator and is resistant to UKS attacks against the Enrollee if the Enrollee performs mutual authentication.

Compromise of static public keys exposes this exchange to active attack. There is no attempt in PQC DPP to protect against maximal exposure (MEX) attacks where an adversary is able to corrupt arbitrary ephemeral outputs. Exposure of the random value used to construct a ciphertext allows an adversary to determine the ephemeral output of a KEM.Decaps() invocation without knowing the secret key. Addressing MEX attacks would require using deterministic encapsulation variants and something like the “twisted PRF” trick in WireGuard.

**Protocol Definition**

Information in brackets—[like this]—is optional and depends on whether the Enrollee wishes to and is capable of performing strong mutual authentication.

The AES-SIV invocations include associated data (AAD) consisting of a vector having two components in the following order: (1) the DPP header, as defined in Table 35 from [1], from the OUI field (inclusive) to the DPP Frame Type field (inclusive); and (2) all octets in a DPP Public Action frame after the DPP Frame Type field up to and including the last octet of the last attribute before the Wrapped Data attribute. The notation is “AAD” in the AES-SIV invocation and the specific contents depend on the message. For example, the first message would have a vector of AAD consisting of the DPP header, followed by a concatenation of the hashes of the two long term identities, the session identifier of the initiator, and the first ciphertext.

 Configurator Enrollee

 <------------------ chirp of SHA256(“chirp” | SPKr)

(eski, EPKi) = KEM.KeyGen()

sidi = rand()

(ct1, sk1) = KEM.Encaps(SPKr)

ck1 = HKDF-Extract(sk1, ct1)

k1 = HKDF-Expand(ck1, “PQC DPP first key”, 64)

msg1 = AES-SIV-enc(k1, AAD, EPKi | icap)

H(SPKr), H(SPKi), sidi, ct1, {msg1} ---------------->

 sidr = rand()

 sk1 = KEM.Decaps(sskr, ct1)

 ck1 = HKDF-Extract(sk1, ct1)

 k1 = HKDF-Expand(ck1, “PQC DPP first key”, 64)

 EPKi | icap = AES-SIV-dec(k1, AAD, msg1)

 (ct2, sk2) = KEM.Encaps(EPKi)

 [ct3, sk3) = KEM.Encaps(SPKi)]

 ck2 = HKDF-Extract(sk1 | sk2 [| sk3], ct1 | ct2 [| ct3])

 k2 = HKDF-Expand(ck2, “PQC DPP second key”, 64)

 rauth = H(SPKr | [SPKi | ] msg1)

 msg2 = AES-SIV-enc(k2, AAD, rcap | rauth)

 <-------------------- H(SPKr), [H(SPKi),] sidi, sidr, ct2, [ct3, ] {msg2}

sk2 = KEM.Decaps(eski, ct2)

[sk3 = KEM.Decaps(sski, ct3)]

ck2 = HKDF-Extract(sk1 | sk2 [| sk3], ct1 | ct2 [| ct3])

k2 = HKDF-Expand(ck2, “PQC DPP second key”, 64)

racp | rauth = AES-SIV-dec(k2, AAD, msg2)

rauth’ = H(SPKr | [SPKi | ] msg1)

if rauth != rauth’

 fail!

k3 = HKDF-Expand(ck2, “PQC DPP third key”, 64)

iauth = H([SPKi | ] SPKr | msg2)

msg3 = AES-SIV-enc(k3, AAD, iauth)

H(SPKr), [H(SPKi),] sidi, sidr, {msg3} ------------------------------->

 k3 = HKDF-Expand(ck2, “PQC DPP third key”, 64)

 iauth = AES-SIV-dec(k3, AAD, msg3)

 iauth’ = H([SPKi |] SPKr | msg2)

 if iauth != iauth’

 fail

Both sides advance to DPP Config exchange using k3 as the AES-SIV key, ke, and ck2 as the key derivation key, bk.

**Follow-On Work**

Bootstrapping of a PQC public key has its own unique challenges and will require more work. Specification of PQC JWS Web Keys and JWS Web Signatures is a work-in-progress at the IETF and that work can be leveraged to construct a PQC Connector conveyed during the DPP Configuration step and used in the DPP Network Access protocol.

A Connector can also be used in the proposed “authentication without signatures” exchange being proposed in the PQC SG/TG.

**References:**

[1] <https://www.wi-fi.org/file/wi-fi-easy-connect-specification>

[2] <https://www.cl.cam.ac.uk/~fms27/duckling/duckling.html>

[3] <https://datatracker.ietf.org/doc/html/rfc5297>