Device Provisioning Protocol Specification

Version 1.0

Document revision history

Version	Date YYYY-MM-DD	Remarks
1.0	2018-04-09	Initial release.

Table of contents

List of tables

List of figures

1 Introduction

This document is the technical specification for Wi-Fi CERTIFIED Easy Connect™, the Wi-Fi Alliance certification program for Device Provisioning Protocol (DPP).This specification defines the architecture, protocols, and functionality for Device Provisioning Protocol devices.

1.1 Scope

The scope of the feature requirements is limited to that defined in this specification. The content of this specification is designed to address the solution requirement areas that include:

1. Supporting public key based identities for all devices. This use of public keys does not require a Central Authority or Public Key Infrastructure.
2. Enabling the use of mobile devices with a user interface to provide a simple user experience during the setup process.
3. Supporting multiple methods for discovery and initiating the secure provisioning of a device with methods that include: In-band, Quick Response codes (QR codes), Near Field Communication (NFC), Label Strings and Bluetooth Low Energy (BLE).
4. Supporting of a scalable third-party authorization mechanism to manage devices in groups to enable a newly configured device to communicate with other members of the group with minimal user interaction.

1.2 References

Knowledge of the documents listed in this section is required for understanding this technical specification. If a reference includes a date or a version identifier, only that specific version of the document is required. If the listing includes neither a date nor a version identifier, then the latest version of the document is required. In the event of a conflict between this specification and the following referenced documents, the contents of this specification take precedence.

[1] Wi-Fi Alliance Device Provisioning Protocol Market Requirements Document, version 1.4, https://groups.wi-fi.org/apps/org/workgroup/wfa-dppmtg/download.php/86833/DPP%20MRD%201.4.docx

[2] IEEE Computer Society, “IEEE Standard for Information Technology– Telecommunications and Information Exchange Between Systems – Local and Metropolitan Area Networks – Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications,” (IEEE Std. 802.11-2016), March 2016

[3] RFC 5297, Synthetic Initialization Vector (SIV) Authenticated Encryption Using the Advanced Encryption Standard (AES), October 2008, https://datatracker.ietf.org/doc/rfc5297/

[4] RFC 5280, Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, May 2008, https://datatracker.ietf.org/doc/rfc5280/

[5] RFC 6090, Fundamental Elliptic Curve Cryptography Algorithms, February 2011 https://datatracker.ietf.org/doc/rfc6090/

[6] ANSI X9.63, Public Key Cryptography for the Financial Services Industry: Elliptic Curve Key Agreement and Key Transport Protocols, 2011.

[7] RFC 5869, HMAC-based Extract-and-Expand Key Derivation Function (HKDF), May 2010, https://datatracker.ietf.org/doc/rfc5869/

[8] Wi-Fi Peer-to-Peer Technical Specification v1.7, https://www.wi-fi.org/members/certification-programs#Wi-Fi Direct https://www.wi-fi.org/members/certification-programs

[9] ITU-T X.509 (10/2012) Information Technology – Open Systems Interconnection—The Directory: Public Key and attribute certificate framework

[10] Wi-Fi Simple Configuration Technical Specification, August 2014, https://www.wi-fi.org/file-member/wi-fi-simple-configuration-technical-specification,

[11] RFC 3339, Date and time on the Internet: Timestamps, July 2002. https://datatracker.ietf.org/doc/rfc3339/

[12] IEEE Computer Society, “Draft Standard for Information technology – Telecommunications and information exchange between systems Local and metropolitan area networks— Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment to IEEE P802.11-REVmcTM/D4.0: Fast Initial Link Setup, Draft 7.0, March 2016.

[13] RFC 7159, The JavaScript Object Notation (JSON) Data Interchange Format, March 2014, https://datatracker.ietf.org/doc/rfc7159/

[14] RFC 4648, The Base16, Base32, and Base64 Data Encodings, October 2006, https://datatracker.ietf.org/doc/rfc4648/

[15] RFC 7515, The JSON Web Signature (JWS), May 2015, https://datatracker.ietf.org/doc/rfc7515/

[16] RFC 7517, The JSON Web Key (JWK), May 2015, https://datatracker.ietf.org/doc/rfc7517/

[17] RFC 7518, JSON Web Algorithms (JWA), May 2015, https://datatracker.ietf.org/doc/rfc7518/

[18] FIPS180-4, "Secure Hash Standard", United States of America, National Institute of Standards and Technology, Federal Information Processing Standard (FIPS) 180-4

[19] RFC 5234, Augmented BNF for Syntax Specifications: ABNF, January 2008, https://datatracker.ietf.org/doc/rfc5234/

[20] U.S. National Institute of Standards and Technology, "DIGITAL SIGNATURE STANDARD", Federal Information Processing Standard FIPS-186-4, July 2013.

[21] RFC 3986, Uniform Resource Identifier (URI): Generic Syntax, January 2005, https://datatracker.ietf.org/doc/rfc3986/

[22] ISO 8601:2004, Data elements and interchange formats -- Information interchange -- Representation of dates and times, December 2004, http://www.iso.org/iso/catalogue_detail?csnumber=40874

[23] draft-harkins-pkex, draft-harkins-pkex05, https://tools.ietf.org/html/draft-harkins-pkex-05

[24] NFC Forum Connection Handover Candidate Technical Specification, December 2015, http://nfc-forum.org/product/nfc-forum-connection-handover-candidate-technical-specification-version-1-4/

[25] RFC 3211, Cryptographic Message Syntax (CMS) Algorithms, August 2002, https://datatracker.ietf.org/doc/rfc3211/

[26] RFC 3602, The AES-CBC Cipher Algorithm and Its Use with IPsec, September 2003, https://datatracker.ietf.org/doc/rfc3602/

[27] RFC 5083, Cryptographic Message Syntax (CMS) Authenticated-Enveloped-Data Content Type, November 2007, https://datatracker.ietf.org/doc/rfc5083/

[28] RFC 5652, The Cryptographic Message Syntax (CMS), September 2009, https://datatracker.ietf.org/doc/rfc5652/

[29] RFC 5958, Asymmetric Key Packages, August 2010, https://datatracker.ietf.org/doc/rfc5958/

[30] RFC 8018, PKCS #5: Password-Based Cryptography Specification, Version 2.1, January 2017, https://datatracker.ietf.org/doc/rfc8018/

[31] RFC 5915, Elliptic Curve Private Key Structure, June 2010, https://datatracker.ietf.org/doc/rfc5915/

[32] RFC 5754, Using SHA2 Algorithms with Cryptographic Message Syntax, August 2010, https://datatracker.ietf.org/doc/rfc5754/

[33] RFC 7906, NSA's Cryptographic Message Syntax (CMS) Key Management Attributes, June 2016, https://datatracker.ietf.org/doc/rfc7906/

[34] RFC 3394, Advanced Encryption Standard (AES) Key Wrap Algorithm, October 2002, https://datatracker.ietf.org/doc/rfc3394/

[35] RFC 2985, PKCS #9: Selected Object Classes and Attribute Types Version 2.0, November 2000, https://datatracker.ietf.org/doc/rfc2985/

[36] Bluetooth Core Specification, v5.0, December 2016, https://www.bluetooth.com/specifications/bluetooth-core-specification/bluetooth5

[37] Supplement to the Bluetooth Core Specification, CSS v7, December 2016, https://www.bluetooth.org/DocMan/handlers/DownloadDoc.ashx?doc_id=421047

[38] The Resurrecting Duckling: Security Issues for Ad-Hoc Wireless Networks, University of Cambridge Computer Laboratory, https://www.cl.cam.ac.uk/~fms27/papers/1999-StajanoAnd-duckling.pdf

[39] RFC 2409, The Internet Key Exchange, November 1998, https://tools.ietf.org/html/rfc2409

[40] Internet Key Exchange (IKE) Attributes, Group Description (Value 4), https://www.iana.org/assignments/ipsec-registry/ipsec-registry.xhtml#ipsec-registry-10

1.3 Definitions and acronyms

1.3.1 Shall/should/may/might word usage

The words shall, should, and may are used intentionally throughout this document to identify the requirements for the Device Provisioning Protocol program.

The word *shall* indicates a mandatory requirement. All mandatory requirements must be implemented to assure interoperability with other Device Provisioning Protocol products.

The word *should* denotes a recommended approach or action.

The word *may* indicates a permitted approach or action with no implied preference.

The words *might* and *can* indicate a possibility or suggestion and should be used sparingly.

1.3.2 Conventions

The ordering of bits and bytes in the fields within information elements, attributes and action frames shall follow the conventions in section 9.2.2 of [2] unless otherwise stated.

The word *ignored* shall be used to describe bits, bytes, fields or parameters whose values are not verified by the recipient.

The word *reserved* shall be used to describe objects (bits, bytes, or fields or their assigned values) whose usage and interpretation will be defined in the future by this specification or by other technical specifications/bulletins. A reserved object shall be set to zero unless otherwise stated. The recipient of a reserved object shall ignore its value unless that

object becomes defined at a later date. The sender of an object defined by this technical specification shall not use a reserved code value.

1.3.3 Abbreviations and acronyms

Table 1 defines the abbreviations and acronyms used throughout this document. Some acronyms are commonly used in publications and standards defining the operation of wireless local area networks, while others have been generated by Wi-Fi Alliance.

Table 1. Abbreviations and acronyms

Acronyms	Definition
AAD	Authenticated Associated Data
AES-SIV	Advanced Encryption Standard - Synthetic Initialization Vector
AKM	Authentication and Key Management
AP	Access Point
ASN.1	Abstract Syntax Notation One
ASP	Application Service Platform
BLE	Bluetooth Low Energy
BSSID	Basic Service Set Identifier
DER	Distinguished Encoding Rules
DPP	Device Provisioning Protocol
ECC	Elliptic Curve Cryptography
ECDH	Elliptic Curve Diffie-Hellman
GHz	Gigahertz
GO	P2P Group Owner
HKDF	HMAC-based Extract-and-Expand Key Derivation Function
ID	Identifier
IE	Information Element
JSON	JavaScript Object Notation
JWS	JSON Web Signature
LSB	Least Significant Bit
MAC	Media Access Control
NDEF	NFC Data Exchange Format
NFC	Near Field Communication
OUI	Organizationally Unique Identifier
P2P	Peer-to-Peer
PKEX	Public Key Exchange
PMK	Pairwise Master Key
PMKID	Pairwise Master Key Identifier
PMKSA	Pairwise Master Key Security Association
PSK	Preshared Key

Acronyms	Definition
QR	Quick Response
RA	Receiver Address
RSN	Robust Security Network
SA	Source Address (IEEE Std 802.11-2007)
SDA	Service Descriptor Attribute
SDF	Service Discovery Frame
SRD	Specification Requirements Document
SRF	Service Response Filter
SSID	Service Set Identifier
STA	Station
TA	Transmitter Address
TLV	Type-Length-Value
UAID	Unique Authenticable Identifier
URI	Uniform Resource Identifier
USB	Universal Serial Bus
WLAN	Wireless Local Area Network
WPA2™	Wi-Fi Protected Access® 2
WSC	Wi-Fi Simple Configuration

1.3.4 Definitions

Table 2 defines the terms that are applicable to this specification.

Table 2. Definitions

Term	Definition
Alternative Carrier	The Wi-Fi communication technology that is to be used for data transfers between an NFC Handover Requester and an NFC Handover Selector.
base64url	Base 64 Encoding with URL and Filename Safe Alphabet as specified in [14].
Bootstrapping key	A public key that is obtained through a bootstrapping method.
Client	A P2P Client or a Legacy Client that is connected to a P2P Group Owner.
Configurator	A logical entity with capabilities to enroll and provision devices for device-to-device communication or Infrastructure communication
Credentials	The information that is required to join a P2P Group as defined in the Wi-Fi Simple Configuration Specification [10].
Connector	Credential information that is provisioned on an enrollee device. The Connector is used by a pair of Enrollee devices to establish a security association using the DPP Network Introduction protocol.
In-band	Data transfer using the WLAN communication channel, including WLAN multiband devices (for example 2.4 GHz, 5 GHz, and 60 GHz).
Initiator	A device that initiates the DPP Authentication protocol. Either the Configurator or the Enrollee can take the role of Initiator.
JWK	JSON Web Key. JSON encoding of keying material as described in RFC 7517 [15].
Legacy Client	A STA that is Wi-Fi CERTIFIED, but not DPP compliant.

Term	Definition
Listen Channel	The channel chosen from the set of commonly available channels in the 2.4 GHz band 8 (1, 6, and 11).
NFC Device	NFC Forum compliant contactless device that supports the following: Initiator, Target, and Reader/Writer. It may also support card emulator.
NFC Handover Requester	An NFC Forum Device that begins the Handover Protocol by issuing a Handover Request Message to another NFC Forum Device.
NFC Handover Selector	An NFC Forum Device that constructs and replies to a Handover Select Message as a result of a previously received Handover Request Message An NFC Forum Tag that provides a pre-set Handover Select Message for reading.
NFC Interface	Contactless interface of an NFC Device.
NFC Tag	NFC Forum compliant contactless memory card that can be read or written by an NFC Device and may be powered by the RF field.
Notification	A mechanism for the DPP stack to inform the application on an event matching criteria given either in Publish or Subscribe.
Operating Channel	The channel on which DPP is operating.
Out-of-Band	Data transfer using a communication channel other than the WLAN
Protocol key	A public key contributed to the DPP Authentication protocol
P2P Client	A P2P Device that is connected to a P2P Group Owner.
P2P Device	Wi-Fi Alliance P2P certified device that is capable of acting as both a P2P Group Owner and a P2P Client.
P2P Device Address	An identifier used to uniquely reference a P2P Device.
P2P Discovery	A capability that provides a set of functions to allow a device to easily and quickly identify and connect to a device and its services in its vicinity.
P2P Interface Address	The MAC address of the P2P interface, an address used to identify a P2P Device within a P2P Group.
Network	A set of interconnection capabilities that share a common access policy.
Persona	An abstraction of the subject of peer authentication and policy enforcement based on the use of a public key.
Peer	A peer is a device provisioned to a network by the Configurator. A peer is able to connect securely to other devices previously provisioned by the Configurator to the same network.
Policy	A list of constraints describing the required attributes of a subject to pass the policy.
Provisioning	Securely enabling a device to establish secure associations with other devices in a network.
Responder	A device that responds to the initiation of the DPP Authentication protocol by the Initiator. Either the Configurator or the Enrollee can take the role of Responder.
Security Domain	An environment comprised of a set of devices that use common security credentials and policies.
Service ID	Service ID in as defined in [1].
Service Name	Service Name as defined in [1]. String matching performed on a Service Name should be case insensitive by converting the single byte values [A-Z] to lower-case before any processing.
Topology	The arrangement in which the nodes of a network are connected to each other and (in some cases) to other networks.

1.3.5 Symbols

[…] A pair of square brackets around one or more fields in a message specification or around variables in an equation signifies that those fields or variables are only present or used in the computation under certain conditions or are optional.

$\{\ldots\}_k$ A pair of curly brackets around one or more fields followed by a variable in a message specification signifies that those fields are encrypted using the value of the variable as the key.

1.4 Architecture

In DPP, public keys are used to identify and authenticate all devices. The private key associated with a public key should be generated within each device and protected from disclosure. Devices use public key cryptographic techniques[1] to authenticate peer devices and establish shared keys for further secure communications. This architecture simplifies the establishment of secure connectivity between devices and provides a foundation for improved usability in provisioning and connecting devices.

1.5 Device roles

The DPP architecture defines the device roles during bootstrapping, authentication, provisioning (configuration) and connectivity (introduction). There are two types of roles, Configurator and Enrollee on the one hand and Initiator and Responder on the other.

A Configurator supports the setup of Enrollees, as shown in Figure 1. The Configurator and the Enrollee engage in DPP bootstrapping, the DPP Authentication protocol and the DPP Configuration protocol. Either Configurator or Enrollee may perform the role of Initiator in the DPP Bootstrapping protocol (for example in PKEX) and in the DPP Authentication protocol. However, only Enrollees initiate the DPP Configuration protocol and the DPP Introduction protocol.

The DPP Authentication protocol requires the Initiator to obtain the bootstrapping key of the Responder as part of a prior bootstrapping mechanism. Optionally, both devices in the DPP Authentication protocol may obtain each other's bootstrapping keys in order to provide mutual authentication.

After the authentication completes, the Configurator provisions the Enrollee for device-to-device communication or infrastructure communication. As part of this provisioning, the Configurator enables the Enrollee to establish secure associations with other peers in the network.

Devices that have been configured by the Configurator are called Peers.

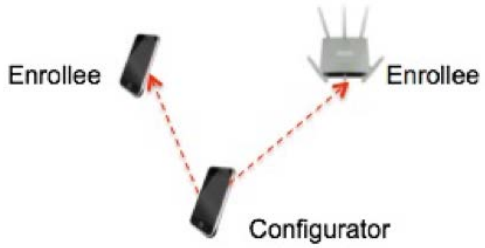

Figure 1. Wi-Fi Device Provisioning roles

1.5.1 Authentication roles

Configurators and Enrollees are involved in the DPP Authentication protocol. Depending on the bootstrapping scenario, either the Configurator or Enrollee may take the role of Initiator or Responder, respectively.

The device that starts the DPP Authentication protocol plays the role of Initiator. The device that responds to an Initiator request plays the role of Responder. The DPP Authentication protocol provides authentication of a Responder to an Initiator, and optionally authentication of the Initiator to the Responder. This assumes that the Initiator has obtained the bootstrapping key of the Responder to perform unidirectional authentication, and both parties have obtained the bootstrapping keys of each other to optionally perform mutual authentication.

For example, a mobile device may act as a Configurator to configure unprovisioned devices acting as Enrollees. An unprovisioned device could include an Access Point or another mobile device. The Configurator mobile device acts as the Initiator when it initiates the DPP Authenticator protocol with the Access Point and the unprovisioned mobile device.

[1] This use of these public keys does not require a Certificate Authority (CA) or Public Key Infrastructure (PKI)

1.5.2 Configurator delegation

A Configurator may delegate its management to another Configurator to share management and provide a backup of the Configurator capabilities, as shown in Figure 2.

Figure 2. Configurator delegation

1.6 Security considerations

1.6.1 Overview

DPP is designed to provide user friendly Wi-Fi setup while maintaining or increasing the security level for the growth and continued penetration of Wi-Fi technology. DPP protocol strives to protect against:

1. *Passive adversaries or eavesdroppers* that could obtain sensitive information by receiving traffic over the Wi-Fi radio interface, either during provisioning or after successful provisioning.
2. *Active adversaries* that could create new networks, add devices to existing networks, or divert devices from the legitimate network to an illicit network.
3. *Active adversaries* that could deny provisioning service without alerting the user.

The following sections describe the security considerations that were taken into account in specifying the DPP protocol.

1.6.2 Threat profile

The DPP protocol has three phases, each of which may be the target of an attack: (1) bootstrapping, (2) authentication and provisioning, and (3) network access.

1.6.2.1 Threats to bootstrapping

Bootstrapping is a form of “original entity authentication” (the step from which all subsequent authentication takes place) and consists of the transfer of a public key credential from a transmitting entity to a relying entity in a manner that allows for establishment of trust in the public key by the relying entity to trust that the public key belongs to the transmitting entity. The threat to bootstrapping is subversion in which a public key under control of the attacker will be transferred to the relying party instead of the actual transmitting entity’s public key being transferred.

Different types of bootstrapping transfer the credential differently and, therefore, provide different attack vectors.

Threats to QR code scanning

A QR code that encodes a public key may be affixed to the transmitting entity or accompany the transmitting entity, for instance as part of a packing list. Subversion of this exchange consists of substituting a QR code that encodes the attacker's public key for the honest transmitting entity’s public key by affixing a QR encoded public key as a sticker on top of the transmitting entity’s QR code sticker, or by exchanging a packing list that contains the attacker’s QR encoded public key, or by replacing the scanning application with a malicious version.

A QR encoded public key that is displayed by a transmitting entity with a suitable user interface is much harder to subvert because it requires physical control of the transmitting entity's user interface. Displaying a QR code on a screen allows the transmitting entity to control who is able to scan it and when. Depending on the Authentication roles (see section 3.2), this can enhance trust in the relying party. An attack against this type of temporal access to a QR code may occur if the transmitting entity leaves the QR code on the screen for an extended period of time, or reuses a QR code for multiple relying entities, or if the attacker replaces the scanning application with a malicious version.

No proof of possession of the private analog to the QR encoded public key is provided with this bootstrapping method, which makes subversion impossible to discover.

Threats to NFC

NFC bootstrapping may be done through negotiated handover between two NFC devices or through static handover using an NFC tag.

Negotiated handover allow devices to exchange bootstrapping information including the bootstrapping key from a device (referred to as "Peer Device A") to another device (referred to as "Peer Device B"). The bootstrapping information can be communicated one way or bidirectional. A threat exists to both technologies due to the fact that Peer Device A will send its bootstrapping key to any device in close proximity. Peer Device B offers the user the opportunity to accept the connection so Peer Device B has the ability to restrict to whom it sends its bootstrapping key, but Peer Device A has no ability to control who receives its key. Another threat exists to both modes through the proximity restriction. Any device that is also within range of Peer Device A and Peer Device B can obtain copies of both device's bootstrapping key. No proof of possession of the private analog to the bootstrapping keys of Peer Device A and Peer Device B is provided with both modes which makes subversion impossible to discover.

Static handover mechanisms allow an NFC device acting as the Initiator to obtain bootstrapping information including the Bootstrapping public key from an NFC Device acting as a Responder or an NFC tag associated with the Responder. Depending on the roles (see section 3.2) this can enhance trust in the relying party. An attack against this type of temporal access to a static NFC tag can be affected by the transmitting entity leaving the NFC tag available for an extended period of time or by re-using a static NFC tag for multiple relying parties. No proof of possession of the private analog to the public key is provided with this bootstrapping method which makes subversion impossible to discover.

Threats to BTLE

Bootstrapping mechanisms that use Bluetooth Low Energy (BLE) technology send bootstrapping information including the bootstrapping key from a device (referred to as "Peer Device A") to another device (referred to as "Peer Device B"). The bootstrapping information can be communicated one way or bidirectional.

A threat exists due to the fact that Peer Device A will send its bootstrapping key to any device in close proximity. Peer Device B offers the user the control of the connection so Peer Device B has the ability to restrict to whom it sends its bootstrapping key but Peer Device A has no ability to control who receives its key. An attack against this type of temporal access to bootstrapping information available through BLE can be effected by the transmitting entity making the bootstrapping information available for an extended period of time or by using the same bootstrapping information for multiple relying parties, in addition to an attack by replacing a scanning application with a malicious version.

Another threat exists to both modes through the proximity restriction. Any device that is also within range of Peer Device A that Peer Device B can obtain copies of both device's bootstrapping key.

No proof of possession of the private analog to the bootstrapping keys of Peer Device A and Peer Device B is provided with both modes which makes subversion impossible to discover.

Threats to proof of knowledge of a shared code (PKEX)

A shared code, key, phrase, or word (hereinafter "code") may be used to encrypt public keys that are transferred in-band over Wi-Fi. The ability to decrypt the public key and use it in a Diffie-Hellman exchange proves that the transmitting and relying entities used the same code. In this bootstrapping technique, each side is both a transmitting entity and a relying entity and they exchange each other's public keys. Since the keys are transmitted in an encrypted manner, a passive adversary cannot observe the public keys being exchanged.

To subvert this bootstrapping method, it is necessary to guess the shared code used to encrypt the public keys. The difficulty in guessing the code is directly proportional to the size of the pool of possible codes from which the particular code was drawn. If the size of the pool of possible codes is X, then the probability is 1/X of a successful guess. This

bootstrapping method is resistant to passive attack and off-line dictionary attack. The only information an adversary learns from an active attack is whether a single guess of the code is correct or not.

In addition, threats against this bootstrapping method are possible if public keys and/or codes are reused. Using the same code with multiple public keys will allow an entity that had successfully bootstrapped public keys in the past to subvert a future bootstrapping by passively observing public keys. Using the same public key with different codes will allow an entity that had successfully bootstrapped public keys in the past to launch an off-line dictionary attack to determine the code, although due to the one-time nature of the code used in this bootstrapping technique an ex post facto attack to determine the code is of dubious value.

Proof-of-possession of the private analog to the exchanged public keys is provided by this bootstrapping method. Therefore, an entity will have proof, with probability of 1-1/X, that the public key it bootstrapped belongs to the peer entity with which it shared the code.

1.6.2.2 Threats to DPP Authentication and Provisioning

The DPP Authentication protocol is exchanged between an Initiator and a Responder. The DPP Provisioning protocol is used by a Configurator to provision an Enrollee. Different attacks are possible depending on who is the Initiator of the DPP Authentication protocol, either a Configurator or an Enrollee.

Threats when mutual authentication is not supported

The DPP Authentication protocol provides optional mutual authentication when both parties have bootstrapped each other's public keys. When mutual authentication is not provided, the Responder does not have the Initiator's public key. In this case, authentication of the Initiator by the Responder is weak and depends on a security model described in [38] that describes how a device will accept authentication from the first entity that attempts it, analogous to a baby duckling that imprints to the first moving object that makes a sound. DPP adds slightly to that model by allowing for knowledge of the Responder's public key to be controlled and to base weak authentication on the Initiator's knowledge of the Responder's public key. A device that can control when and how its public key is bootstrapped (see section 3.2.1) can weakly authenticate an entity that knows its public key.

Additional attacks are possible when mutual authentication is not employed by the Responder.

Regardless of whether mutual authentication is employed or not, an additional denial of service attack is possible when the attacker sees the hash of the Responder's public key and constructs bogus DPP Authentication Request frames to flood the Responder by inducing it to perform Diffie-Hellman exponentiations.

Threats when the Enrollee is the Initiator

The attacker in this scenario is posing as an Enrollee that wants to be provisioned by the Configurator and obtain a credential for accessing the network/services controlled by the Configurator.

When mutual authentication is not employed, the Configurator does not strongly authenticate the Enrollee. The Configurator is forced to give out a credential for network/resource access to an entity whose identity has not been strongly determined. The potential for successful attack depends on the amount of control the Configurator has, and has exerted, over how its public key gets bootstrapped.

When mutual authentication is employed, the Configurator will have the Enrollee's public key and will strongly authenticate the Enrollee. Credentials for network/resource access will only be given to entities whose identity can be determined based on the bootstrapping method used.

Threats when the Configurator is the Initiator

An attacker in this scenario wants to provision a device and put the device on its network, effectively controlling it. This is a form of resource hijacking.

When mutual authentication is not employed, the Enrollee does not strongly authenticate the Configurator. The Enrollee receives a credential for network/resource access and a signing key to use for validating other parties' credentials from an entity whose identity has not been determined. The Enrollee now obtains network access through the network controlled by the Configurator. The potential for successful attack depends on the amount of control the Enrollee has, and has exerted, over how its public key gets bootstrapped.

When mutual authentication is employed, the Enrollee will have the Configurator's public key and will reject provisioning by a Configurator that cannot prove possession of its private key. Credentials for network/resource access and a signing key will only be accepted from an entity whose identity can be determined based on the bootstrapping method used.

1.6.2.3 Threats to network access

An attacker in this scenario wishes to obtain elevated privileges. All provisioned devices will have credentials issued by the Configurator and have a copy of the Configurator's signing key to validate each other's credentials.

If a Configurator is willing to engage in the DPP Authentication protocol as both an Initiator and Responder, an attacker may take advantage of the authentication asymmetry and interact with other peers who may have strongly or mutually authenticated the Configurator and assume the attacker did as well.

This threat may be mitigated if the Configurator provides the same level of authentication to all peers it has provisioned, always require mutual authentication, and use the same bootstrapping methods for different Enrollees.

1.6.3 Trust model

Trust is "the belief in the reliability and truth of information or in the ability and disposition of an entity to act appropriately, within a specified context" according to ITU-T X.509, [10].

All entities in DPP shall use a public/private key pair in the DPP Authentication protocol. Trust needs to be placed in the public keys used in DPP and in the expected results of running the DPP protocols.

1.6.3.1 Trust in bootstrapped public keys

In DPP, trust in a public key is the assurance that the private key is known only by the entity that is presenting or responding to the public key. Trust in a public key is obtained through bootstrapping.

The trust placed in a public key obtained through a bootstrapping method depends on the potential for the bootstrapping method to be subverted or attacked. Because an absolute measure of public key trust is not possible, it is useful to think of trust relatively and note that some bootstrapping methods provide more trust than others (meaning they have less probability of being subverted and have different abilities to detect subversion.).

Each DPP entity will determine the level of trust it places in each public key it bootstraps and will not use a public key if it is not able to obtain suitable trust.

1.6.3.2 Trust in the DPP Authentication and Provisioning protocol

The DPP Authentication protocol has a decentralized architecture with no central authority to coordinate or control authentication. It therefore relies on a direct trust model. Each side of the entities shall perform the necessary validation of the other to meet the requirements of the exchange.

When DPP entities employ mutual authentication to communicate between each other the level of trust that each side is communicating with, the correct entity is inversely proportional to the probability of the employed bootstrapping method being subverted.

Regardless of which entity initiates the DPP Authentication protocol, the Enrollee trusts that the Configurator only issues credentials to devices that have been authenticated at least as strongly as it authenticated the Enrollee, that the Configurator only issues credentials for exactly the same purpose as for that it issued to the Enrollee, and that the Configurator has sole possession of its private signing key. The Configurator trusts that the public key inside the credential issued to the Enrollee belongs to the Enrollee.

When there is no mutual authentication, the Responder weakly authenticates the Initiator and trusts it due to ownership, proximity, or temporal access. If the Initiator gained knowledge of the Responder's public key, the Responder can take into account the susceptibility of the bootstrapping method it supports to being subverted. Due to the asymmetric nature of the Configurator/Enrollee trust relationship, the party that imprints has particular trust considerations depending on whether the party is the Configurator or Enrollee. These considerations balance the due diligence the imprinting party needs to perform to ensure its trust is not misplaced against the liability of wrongfully trusting a peer. This is known as achieving a trust balance and is highly use-case dependent.

Imprinting of Enrollee

When the Enrollee responds in the DPP Authentication protocol without mutual authentication, it is accepting imprinting by an entity it cannot strongly authenticate. Part of the DPP Provisioning protocol is the assertion by the Configurator of an Enrollee-specific credential signed by the Configurator and the Configurator's signing key used to validate the credentials of other entities provisioned by the Configurator. Depending on the liability in the specific use case, an Enrollee may refuse to engage in a bootstrapping technique that does not meet the required due diligence necessary to achieve a trust balance. The Enrollee should achieve trust that the Configurator imprinting the Enrollee is legitimate.

For example, a hot water heater can have wireless capabilities to perform certain "smart grid" applications. This requires connectivity to a home network and the DPP protocol can be used to provision an AP to create the necessary BSS and provision stations, such as a "smart grid"-enabled hot water heater, to connect to the AP. The liability in this case is a hot water heater that can be manipulated by an attacker or used by an attacker to gain access to other devices on the home network. Given that hot water heaters are not mobile or portable, the due diligence of bootstrapping via scanning a QR code printed on a label physically affixed to the hot water heater or printed on packing materials should achieve a trust balance. A hot water heater that has never been provisioned is able to trust the first Configurator that proves it knows the hot water heater's public key due to the nature of installation of hot water heaters. It can, therefore, gain trust that the Configurator is controlled/owned by the installer of the hot water heater and is, therefore, legitimate.

Imprinting of Configurator

When the Configurator responds in the DPP Authentication protocol without mutual authentication, it is accepting imprinting of a device it cannot authenticate and will therefore provide a credential to access a protected network/resource during the DPP Provisioning protocol that it has not strongly authenticated. Depending on the liability of the particular use case, a Configurator may only engage in bootstrapping techniques that meet the required due diligence, that is it will only accept authentication (and proceed to provisioning of the Enrollee) if it is able to achieve a trust balance.

For example, a badge printing device that issues temporary badges to visitors to an enterprise can additionally have wireless capabilities This device, acting as a Configurator, can issue a dynamic QR code for visitors as well as a badge. Visitors can scan their personal QR code and engage in DPP with the device in order to be provisioned for guest network access. The liability in this case is an attacker obtaining unauthorized network access. To protect against such attacks, the device enforces access control for the dynamic QR codes it generates by producing them along with the temporary visitor badge. Then an Enrollee that proves possession of this dynamic public key can be trusted to have obtained the QR code as part of the badging process and, therefore, be legitimate. Once the bootstrapped public key has been used by the Enrollee, the device can destroy it along with the private key. The device can trust the results of the DPP Authentication protocol because the due diligence, in the form of access control and one-time usage of public keys, for this bootstrapping method exceeds the liability the device is under—it has achieved a trust balance and can issue the credential necessary to permit network access.

1.6.3.3 Trust in the DPP Network Access protocol

The DPP Network Access protocol relies on a transitive trust model due to the signed nature of Configurator-issued credentials. A device that has a Configurator-issued credential demonstrates that the Configurator provisioned it, that the Configurator established trust in it. An Enrollee trusts that the Configurator that issued its credential authenticates all other devices to which it issues credentials at least as strongly. An Enrollee also trusts that the Configurator does not issue credentials, signed by the same signing key, for different purposes—an Enrollee assumes that another peer with a valid Connector got that Connector for the same purpose as the Enrollee. This allows for transitive trust to be established:

- Alice trusts Configurator Charlie
- Bob proves to Alice that Charlie trusts Bob
- Alice trusts Bob

2 DPP protocol usage

2.1 Overview

The DPP protocols are typically exchanged between one pair of devices, where one device takes on the role of Configurator and the other device takes on the role of Enrollee. Either device may initiate the DPP protocol. Roles are assumed by the Initiator and Responder based upon the type of network that DPP is being used to set-up.

2.2 Infrastructure setup and connectivity

A Configurator may be used to setup an infrastructure network consisting of one or more APs. A Configurator may setup clients (STA devices) and APs in any order and give each of the devices appropriate configuration information to support subsequent discovery and connectivity to the infrastructure network.

2.2.1 AP configuration

The configuration information provided to an AP may include:

- The SSID for the BSS
- Fields to be carried in Beacon and Probe Response frames to designate the connectivity policy
- Operating channel or band information
- One or more Connectors generated by the Configurator for the authorized connectivity
- An optional WPA2-Personal passphrase(s) or PSK(s) for the BSS to support legacy devices

2.2.2 STA configuration

The configuration information provided to a STA may include:

- The SSID for the BSS
- Fields to be used to identify networks with the supported connectivity policy
- One or more Connectors generated by the Configurator for the authorized connectivity
- An optional WPA2-Personal passphrase or PSK for the BSS to support legacy devices

2.2.3 Infrastructure connectivity

The Configurator provisions both AP and non-AP STA Enrollees to establish a trusted and secure connection.

To provide a scalable method of enabling new Enrollees to connect to the AP without having to contact the AP each time, the Configurator provisions Connectors (see section 4.2) on each Enrollee. A Connector is a tuple of a Group Identifier, a Network Role, and a Network Access Key, all signed by the Configurator. The Group Identifier may indicate a specific peer or a wildcard that indicates all peers.

2.2.4 Message flows for infrastructure connectivity

The message flows for infrastructure connectivity are given in Figure 3 and Figure 4. Figure 3 shows a message flow where a Configurator provisions a STA Enrollee with a Connector. The DPP Bootstrapping protocols are described in detail in section 5. The DPP Authentication protocol is described in section 6.2, and the DPP Configuration protocol is described in section 6.4. The STA Enrollee then uses the Connector to establish a secure connection to an AP via the Network Introduction protocol as shown in Figure 4. The DPP Network Introduction protocol is described in section 6.4.

The DPP Authentication protocol requires either the Enrollee or the Configurator to take on the role of Initiator. In the example message flow shown in Figure 3, the Configurator takes on the role of Initiator and obtains bootstrapping information for the Enrollee. The bootstrapping information includes the public bootstrap key as well as a Global Operating Class and a Channel Number list [2] that the STA Enrollee listens on to perform the DPP authentication protocol.

The Initiator issues DPP Authentication Request frames on channels in the channel list and waits for a response from the Responder. After successfully receiving a response, the Initiator validates the result and transmits a DPP Authentication

Confirm frame to complete DPP Authentication. After successful completion of these frame exchanges, a secure channel between the Initiator/Configurator and Responder/Enrollee is established.

The STA Enrollee initiates the provisioning phase by transmitting a DPP Configuration Request frame, and is provisioned with configuration information in a DPP Configuration Response frame. After successfully receiving the DPP Configuration Response frame, the Enrollee is provisioned with the information required to establish secure access to the AP.

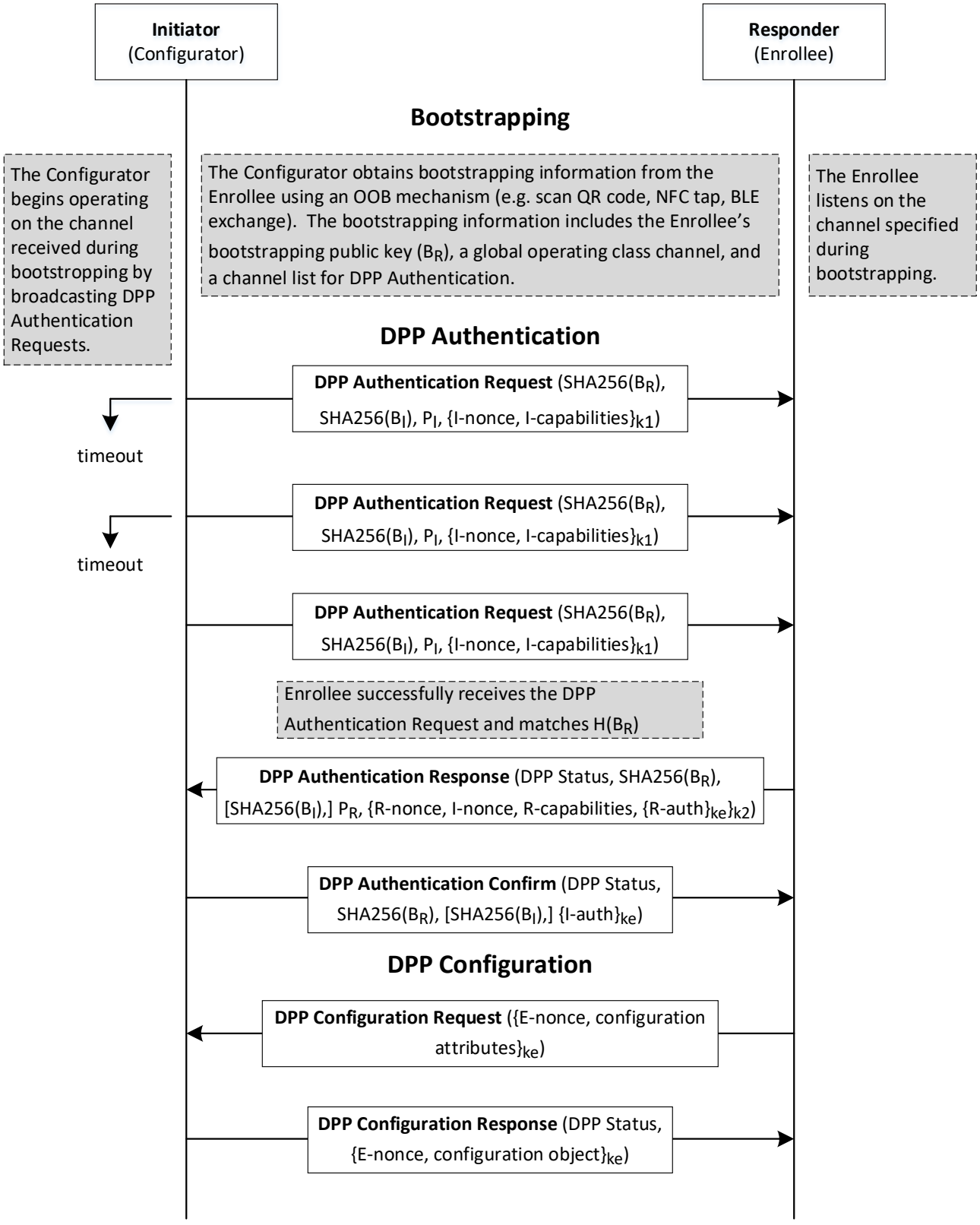

Figure 3. DPP message flow for DPP Provisioning of an STA Enrollee

If the STA Enrollee is provisioned with a Connector as credentials, it discovers the AP, transmits a Peer Discovery Request frame and then waits for a Peer Discovery Response frame. Upon successful validation of the Peer Discovery frames, the STA and AP mutually derive a PMK and PMKID, and the STA follows the standard IEEE 802.11 procedures.

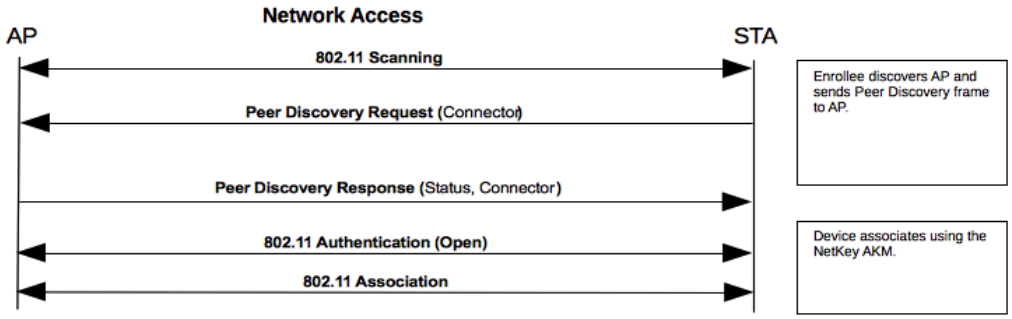

Figure 4. Example message flow for network access using a Connector

Alternatively, if the STA Enrollee is provisioned with PSK or PSK Passphrase credentials to allow it to connect to a legacy AP, the STA Enrollee will use the configuration to discover and associate to an AP using IEEE 802.11 and WPA2-Personal network access procedures. The DPP Bootstrapping, DPP Authentication, and DPP Configuration procedures are the same as the example shown in Figure 3.

2.3 Wi-Fi Direct

NOTE: THIS FEATURE HAS NOT BEEN TESTED IN THE DEVICE PROVISIONING PROTOCOL (DPP) CERTIFICATION PROGRAM.

Wi-Fi Direct [8] provides a mechanism to establish device-to-device connection. When using DPP, a P2P Device that desires to join a P2P network shall support the following functionalities:

- P2P functions (such as P2P Discovery, Group Formation)
- DPP functions:
  - QR code bootstrapping
  - In-band bootstrapping (using PKEX Protocol)
  - Optionally support NFC bootstrapping

A P2P network is composed of a P2P Group Owner, and at least one P2P client or legacy client. A P2P connection is established by going through two stages: (1) P2P Discovery and (2) Group Formation and Provisioning. P2P Discovery may be performed using an in-band or out-of-band mechanism to acquire information about the peer device. Group Formation establishes the roles of the P2P Devices in the P2P Group and assigns ownership of the group to a P2P Device with “AP-like” capabilities and can provide BSS functionalities. DPP may be used to provision the network, in which case the P2P Group Owner acts as Initiator and the P2P Client will act as Responder.

Figure 5 shows the high-level configuration process of Wi-Fi P2P with DPP as provisioning protocol using the QR code bootstrapping method.

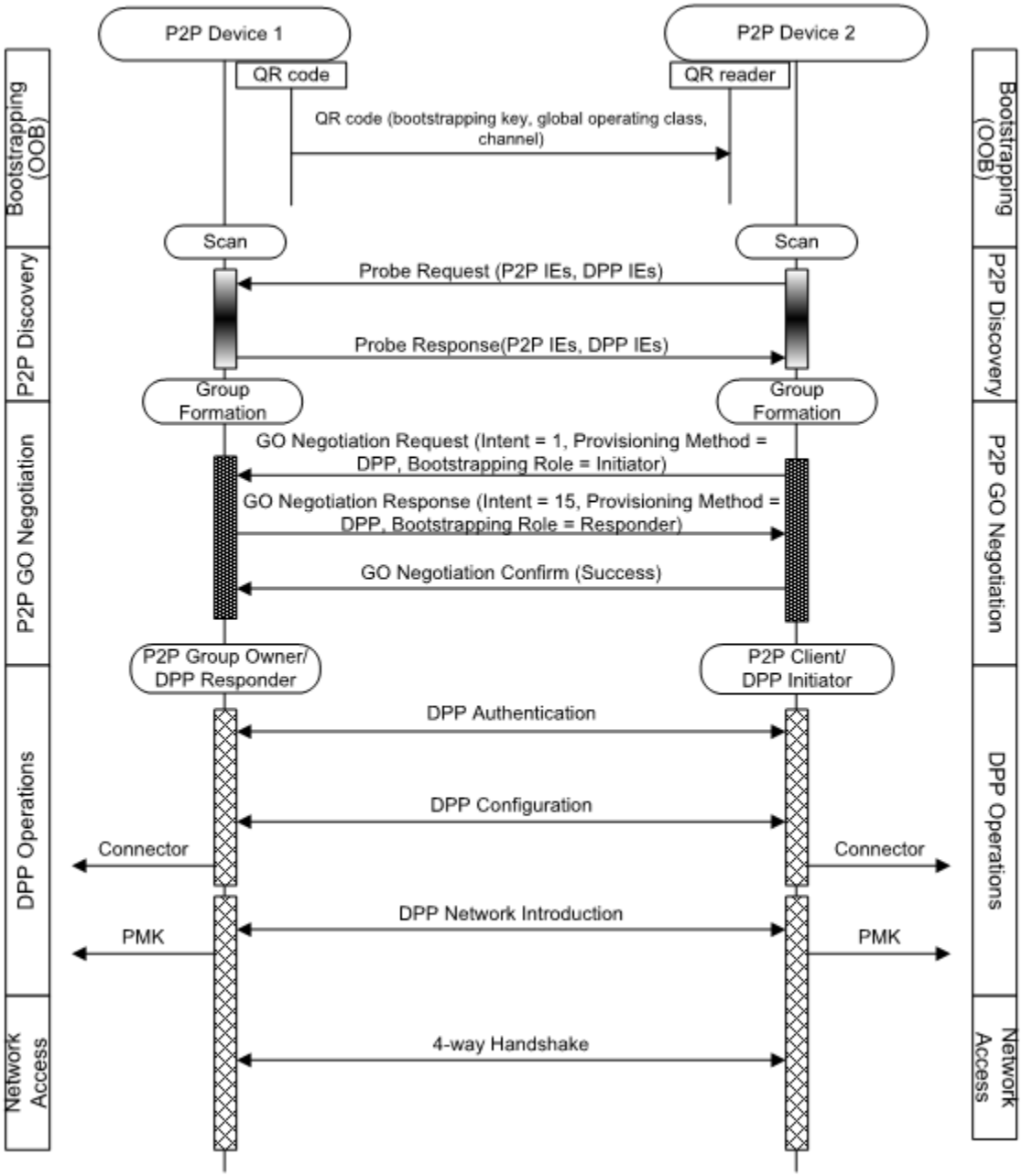

Figure 5. Wi-Fi P2P provisioning via DPP using QR code and In-band P2P Discovery

2.3.1 Establishing a P2P group using DPP

The following steps are required to establish a P2P group using DPP.

Step 1: Bootstrapping of trust

The public keys of the devices are transferred from one device to another, which places trust that the public key belongs to the peer device. This may be performed using either the out-of-band (OOB) mechanism for devices with limited capability or using in-band bootstrapping (PKEX frames) for devices with a user interface.

Step 2: P2P discovery

P2P Discovery information is acquired either in-band or OOB via Probe Request and Probe Response. Alternatively, P2P Discovery information may be included in bootstrapping of OOB data. P2P Discovery is further discussed in section 2.3.1.1.

Step 3: P2P Group Owner negotiation

A P2P device's role and group's ownership and characteristics are determined during P2P Group Owner negotiation.

Step 4: DPP Discovery

DPP Discovery is triggered by a P2P Group Owner acting as an Initiator to ensure that the P2P Client to be provisioned via DPP is still active or ready for provisioning.

Step 5: DPP Authentication

DPP Authentication verifies the public key acquired during the bootstrapping phase via encrypted action frames.

Step 6: DPP Configuration

DPP Configuration verifies the attributes of the device and provides the Connector to be used to gain access to the network.

Step 7: DPP Network Introduction

DPP Network Introduction protocol allows communication to devices using the same Connector broadcasted by the provisioned client (refer to section 6.4).

2.3.1.1 P2P Discovery

P2P Discovery procedure

P2P Discovery enables a device to find a P2P Device to which a connection may be attempted. There are two methods for device discovery:

1. In-band discovery
    a. Follow the existing Wi-Fi P2P procedure (Scan and Find Phase)
    b. Include the Provisioning Method attribute in the Probe Response to establish that DPP will be used as the provisioning method (refer to section 4.1 of [8]).
2. Out-of-Band discovery

    Include P2P Discovery information in the OOB data (similar with NFC Out-of-Band Device Discovery, refer to section 4.4 of [9]).

Group formation procedure

P2P Group formation is used to form a P2P Group to determine the Group Owner, exchange the credentials, and identify the group's characteristics (for example, Persistent P2P group or Temporary P2P Group). There are two phases in P2P Group Formation:

1. Group Owner Negotiation
    a. Follow the existing Wi-Fi P2P procedure (refer to section 3.1 of [8]).
    b. After role negotiation finishes, the P2P Group Owner shall serve as the Initiator, and the P2P Client shall become the Responder during DPP provisioning.
2. Provisioning
    a. P2P Group Owner as the Initiator starts DPP provisioning
    b. Bootstrapped OOB data is verified
    c. Connector is provided to P2P Client to gain access over the P2P Group
    d. Connector acquired during DPP provisioning is used during the Network Introduction protocol

2.3.2 P2P Group operation

P2P Group operation is the same as the existing Wi-Fi P2P or BSS operation. This includes:

- Group ID
- SSID begins with “DIRECT-“
- Operating channel
- Usage of P2P interface address
- Connecting to a P2P Group using WPA2-PMK Security

3 Security

3.1 Properties

The DPP Authentication protocol provides authentication of a Responder to an Initiator and optionally authentication of the Initiator to the Responder, and generates a shared key between the Initiator and Responder that is unpredictable and pseudorandom. The DPP provisioning protocol allows an Initiator to provision a Responder to facilitate a secure connection to a network. A consequence of successful DPP provisioning is trusted public keys that allow for a unique, pairwise secret key to be established for every connection on the network.

3.2 Public key cryptography

3.2.1 Supported public key cryptosystem

The only cryptosystem supported by DPP is elliptic curve cryptography (ECC). All elliptic curves used in DPP shall be over fields with a characteristic greater than three, as defined in RFC 6090 with a co-factor of one (1) [5]. DPP devices shall support elliptic curve Diffie-Hellman (ECDH) and have an ECDH Identity key. All network access provisioning keys, bootstrapping keys, Signature and protocol keys shall use ECC [9]. All ECDH operations shall be performed with compact output per RFC 6090 [5]. For signed introduction, the device shall support Elliptic Curve Digital Signature Algorithm (ECDSA).

3.2.2 Notation

The following notations are used in this specification.

The elliptic curve E is a group of prime order p, with a generator G. Scalar values are indicated with lowercase and points on the curve with uppercase. The addition symbol "+" when used on two scalars is arithmetic addition. When used on two elements, the addition symbol is point addition as defined by RFC 6090 [5]. The subtraction symbol "–" when used on two elements is addition of an element and the inverse of another element. The multiplication symbol "*" when used on two scalars is arithmetic multiplication. When used on a scalar and an element, the multiplication symbol indicates repeated addition of the element with itself by the number of times indicated by the scalar.

Each point on the curve is an element of E. Key pairs consist of a scalar private key and a corresponding public key whose value is the generator of the curve multiplied by the private key. For example:

Pub = priv * G

Each point on the curve has (x,y) coordinates. A complex element may be converted into a scalar by taking its x-coordinate and ignoring the y-coordinate. For example:

val = Pub.x

SHA256() is the SHA256 hash function as specified in FIPS180-4, [18].

H() is a hash function used as a random function. The hash algorithm selection is described in section 3.2.3.

HKDF [7] is used for key derivation with the following notation:

key = HKDF(salt, info, ikm)

= HKDF-Expand(HKDF-Extract(salt, ikm), info, len)

where salt, ikm, HKDF-Extract(), and HKDF-Expand() info are as defined in RFC 5869 [8], and len, the length of the resulting key is equal to the digest length of the hash function used with HKDF.

<> passed as a salt to HKDF indicates the salt-less invocation of HKDF.

Truncate-n(x) returns the left-most n bits of x if the length of x is greater than n. If the length of x is less than or equal to n, Truncate-n(x) returns x.

Ceiling(x) returns the smallest integer greater than or equal to x.

AES-SIV is used to wrap cleartext into ciphertext, with the following notation:

ciphertext = {cleartext}k

Concatenation is denoted with the following notation:

result = A | B

3.2.3 Cryptographic suites

DPP provides for cryptographic agility by indicating the particular cryptographic suite of primitives to use in DPP messages. Currently only cryptographic suite 1 is defined. Cryptographic suite 1 consists of the SHA2 family of hash algorithms and AES-SIV.

NOTE: Other cryptographic suites can be defined in the future.

For interoperability purposes, all devices supporting cryptographic suite 1 shall also support the NIST P-256 elliptic curve, see FIPS PUB 186-4 [20]. All devices shall also support SHA256 [18] and AES-SIV-256 [3]. Table 3 defines the cryptographic primitives used to support other elliptic curves.

The DPP protocol design binds the keys used in bootstrapping, authentication, and data connection in a manner that mandates a single elliptic curve is selected for all devices in any given network. DPP devices shall use the NIST P-256 elliptic curve for their bootstrapping and protocol keys for interoperability unless a network is explicitly configured to be used only by devices capable of using another common curve.

With cryptographic suite 1, the particular hash algorithm to use with the DPP protocols as H() and in HKDF(), the length of the key for AES-SIV, and the size of the nonces used in the DPP Authentication protocol all depend on the length of the prime, p, used in constructing the elliptic curve as shown in Table 3.

Table 3. Key and Nonce Length dependency on Prime Length

Length of prime	Hash algorithm	AES-SIV key length	Nonce length
len(p) ≤ 256	SHA256	256	128
256 < len(p) ≤ 384	SHA384	384	192
len(p) > 384	SHA512	512	256

3.2.4 Point representation

Points on the elliptic curve are transmitted in DPP protocol messages by converting both coordinates of the point into a single octet string using the Element-to-Octet-String conversion technique of section 12.4.7.2.4 of [2]. The resulting octet string, whose length is twice the length of the prime used in constructing the elliptic curve, is transmitted as the value of the appropriate DPP attribute (see section 8.1.1). However, points on the elliptic curve are represented in uncompressed form in JSON Web Keys in the Connector and the DPP Configuration object.

Upon receipt of an elliptic curve point in a DPP protocol message, the recipient reconstructs the point by converting the octet string into a point using the Octet-String-to-Element conversion technique of section 12.4.7.2.5 of [2]. All received points shall be verified to be on the elliptic curve by checking that the received coordinates satisfy the equation of the curve, $y^2 = x^3 + ax + b$ has a solution (x,y) or (x,-y), excluding the “point-at-infinity”.

Compact output of an ECDH operation is a scalar value representing the x-coordinate of the point (see section 3.2.2). When used as an input to a function, the scalar is first converted into an octet string by treating it as an integer and converting it to a Ceiling(len(p)/8) octet string using the Integer-to-Octet-String conversion technique of section 12.4.7.2.2 of [2].

4 Data structures

4.1 Public keys

Bootstrapping keys shall only be used for authentication of the DPP exchange that performs configuration and enrollment.

Bootstrapping keys are the ASN.1 SEQUENCE SubjectPublicKeyInfo from [4]:

```
AlgorithmIdentifier ::= SEQUENCE {
       algorithm          OBJECT IDENTIFIER,
       parameters         ANY DEFINED BY algorithm OPTIONAL }

SubjectPublicKeyInfo ::= SEQUENCE {
       algorithm             AlgorithmIdentifier,
       subjectPublicKey      BIT STRING }
```

where the *subjectPublicKey* is the compressed format of the particular public key per [6]. For the DPP protocols, the algorithm OBJECT IDENTIFIER is the object ecPublicKey (1.2.840.10045.2.1) and the OPTIONAL parameters shall be present and shall be the OBJECT IDENTIFIER that defines the elliptic curve, for instance the object prime256v1 (1.2.840.10045.3.1.7).

Distinct keys, called network access provisioning keys, shall be used by peers for network access/formation. Network Access Provisioning and Configurator Signature keys shall be encoded in JSON Web Key (JWK) format [15]. Points on the elliptic curve are represented in uncompressed form in JSON Web Keys.

4.2 Connectors

Configurators may present enrolled devices with Connectors, which are signed introductions that allow the Enrollee to obtain a trusted statement listing the other devices on the network that the Enrollee is permitted to communicate with. A Connector is the JSON Web Signature (JWS) Compact Serialization of a JWS Protected Header (describing the encoded object and signature), a JWS Payload, and a signature. The JWS Compact Serialization is a base64url encoding of each component, with components separated by a dot, “.”.

The JWS Protected Header is a JSON object that describes:

- the type of object in the JWS Payload specified as "dppCon",
- the identifier for the key used to generate the signature (specified in section 6.3.5.2),
- and the algorithm used to generate a signature.

The "dppCon" mediatype is the data structure defined in section 6.3.5, Table 13. The contents of the "dppCon" object represent the JWS Payload of the Connector.

The supported algorithms are given in [17]. All devices supporting DPP shall support the ES256 algorithm. Key and nonce lengths shall be as specified in Table 3. The curve used for the signature may be different from the one used in DPP Bootstrapping and DPP Authentication protocols.

The signature is performed over the ASCII representation of the concatenation of the base64url encodings of both the JWS Protected Header and the JWS Payload, separated by a dot, “.”, see section 5.1 of [15].

The data passed to the signature algorithm is:

```
ASCII(base64url(UTF8(JWS Protected Header)) | ‘.’ |
base64url(JWS Payload))
```

where ASCII(s) is the ASCII representation of string “s” and UTF8(s) is the UTF8 representation of the string “s”.

If “sig” is the result of the signature, the Connector is then:

```
base64url(UTF8(JWS Protected Header)) | ‘.’ |
```

```
base64url(JWS Payload) | '.' |
base64url(sig)
```

The Connector authorizes the entity that possesses the *netAccessKey* to connect to a class of devices that support a common set of attributes. The Configurator vouches for the two peers to allow them to make network connections on the network configured with complementary attributes. The *netAccessKey* is a public key, signed by the Configurator that a device uses to establish secure connectivity to another device.

Two devices that are provisioned to the same network verify the origin of their Connectors by using the verification scheme in ECDSA [10] section 6.4.2. If the verification succeeds, the devices may use their network keys to agree on a PMK and a PMKSA, which is followed by traditional WPA2 mechanisms for Wi-Fi communication.

When two devices successfully validate received Connectors, —for example a STA is provisioned with a Connector group attribute matching the AP's *groupId* and *netRole* of the STA or Client, and the AP is provisioned with a Connector group attribute containing a matching *groupId* and a *netRole* of the AP or GO to allow connection to all devices on the network— a shared PMK may be derived by both parties using the other party's *netAccessKey* and the private analog to its own *netAccessKey*. This PMK may be used with, for example, the 4-way Handshake from [2].

4.3 DPP Configuration object

A Configurator provisions an Enrollee with information to discover a network as well as credentials to establish secure access to the network.

The DPP Configuration object contains the following nodes:

- Wi-Fi Technology: the Wi-Fi technology that is being provisioned
- DPP Discovery: Information provided for network/device discovery
- DPP Credential: Credential information for network access

4.3.1 Wi-Fi Technology

The Wi-Fi Technology node identifies the Wi-Fi technology of the policy that is to be provisioned within the Enrollee device. It may have one of the following values:

- DPP Configurator, if the enrollee is provisioned as a Configurator
- Infrastructure, if the enrollee is provisioned as either a STA or an AP
- Peer to Peer (P2P)[2], if the enrollee is provisioned as a P2P Device

The Configurator shall set the value of this node depending on the Wi-Fi Technology that is in operation between the Enrollee and the Configurator.

The Enrollee reads the value of this node to determine the type of provisioning system to use.

4.3.2 DPP Discovery

The DPP Discovery node contains optional operating/discovery information such as SSID, operating channel and operating band.

The Configurator sets the value of this node to the values of a Wi-Fi network (for example SSID and channel) that are to be provisioned within the client device.

The Enrollee reads the value of this node and provisions the Wi-Fi network information.

4.3.3 DPP Credential

The DPP Credential node contains the credential information provisioned in the Enrollee to obtain secure network access.

2 NOTE: THIS FEATURE HAS NOT BEEN TESTED IN THE DEVICE PROVISIONING PROTOCOL (DPP) CERTIFICATION PROGRAM.

The credential information included in the configuration is dependent on the value of the AKM parameter. The AKM parameter may either be set to "dpp" to indicate that a Connector is being provisioned or "psk" if a legacy passphrase or pre-shared key is being provisioned. The DPP Credential may contain a Connector, C-sign-key, legacy PSK or passphrase.

4.3.3.1 Connector

The Connector is present when the AKM is set to "dpp". The Configurator provisions the Connector on the Enrollee.

The Configurator derives and sets the Connector in the DPP Configuration object.

The Enrollee reads the Connector and provisions the configuration.

4.3.3.2 C-sign-key

The C-sign-key is present when the AKM is set to "dpp". The C-sign-key is the base64 encoded public signing key of the Configurator.

The Configurator derives and sets the C-sign-key in the DPP Configuration object.

The Enrollee reads the C-sign-key and provisions the configuration.

4.3.3.3 Legacy PSK or passphrase

The Legacy PSK or Passphrase is present when the AKM is set to "psk". The PSK or Passphrase is either a string of 8-63 characters or a string of 64 hex values. It is used when the Configurator provisions legacy WPA2 Personal credentials on the Enrollee device. This configuration element is present when the AKM is set to a legacy WPA2-Personal value.

The Configurator sets the value of the PSK or Passphrase in the DPP Configuration object.

The Enrollee reads the value of the PSK or Passphrase and provisions the configuration.

4.3.3.4 Expiry

The expiry defines the expiry date for the connector information on the DPP client. After the expiry date, the information is no longer valid and may be updated by the Configurator when a further connection between an Enrollee and a Configurator is established.

The date and time is formatted as described in [22].

The Configurator sets the value of expiry to a credential expiry date in the DPP Configuration object.

The Enrollee reads the value of provisions, the expiry date, and provisions the configuration. If there is no expiry value, then it is assumed that there is no expiry date for the information.

The Enrollee uses the expiry date to indicate whether the credentials received by the Configurator are valid. The Enrollee compares the expiry to the current time to evaluate the validity of the credentials. The Enrollee shall not use expired credentials.

5 Bootstrapping of trust

5.1 Overview

ECDH Identity keys shall be transmitted out-of-band between the entities prior to performing the DPP protocol. The manner in which the Identify keys are transferred shall allow the recipient to gain a certain amount of trust that the public key obtained belongs to the transmitting device. A device shall use a distinct pair of keys for each bootstrapping method it supports. For each such pair, the public key is the bootstrapping key for the specific bootstrapping method.

A table mapping the hash of the bootstrapping key to the pair associated to the specific bootstrapping method shall be maintained by a device. This table shall be called the bootstrapping table, and its entries shall be called bootstrapping entries.

Network access provisioning keys do not need bootstrapping because they shall only be transferred through DPP protocol messages after authentication.

Regardless of the technique used to bootstrap trust, the received public key shall be validated as specified in section 3.2.4. Once validated, all bootstrapping keys shall be stored in a canonical representation as DER-encoded ASN.1 SubjectPublicKeyInfo constructed as specified in section 4.1. If the bootstrapping technique does not exchange keys in this format, the keys shall be put into this format before being used in the DPP Authentication protocol.

5.2 Bootstrapping information

Regardless of the bootstrapping method, the public bootstrapping key in the format of ASN.1 SEQUENCE SubjectPublicKeyInfo, as described in section 4.1, shall be included to facilitate DPP Authentication.

The device may optionally include additional bootstrapping information that may be used in authenticating and provisioning the peer. While these fields are optional, devices are recommended to include them to allow optimized device discovery at the beginning of the DPP Authentication exchange. For example, the device may include:

- A list of global operating class/channel pairs
- The MAC address

If the global operating class/channel list is included in the bootstrapping information, the device indicates that it shall be listening on one of the listed channels for other devices to initiate the DPP Authentication exchange. If this list is not included, the device does not provide any guidance on which channel it is listening on and the Initiator shall iterate over all available channels. Iteration over a large number of channels adds significant extra delay in the DPP Authentication exchange; therefore, devices using QR Code bootstrapping are recommended to include a single channel or at most a short list of possible channels in the bootstrapping information.

5.2.1 Bootstrapping information format

The bootstrapping information is transmitted as a URI according to RFC 3986 [21] or as a Bootstrap Information attribute in PKEX and is formatted by the dpp-qr and pkex-bootstrap-info ABNF rules:

```
dpp-qr = “DPP:” [channel-list “;”] [mac “;”] [information “;”] public-key “;;”
pkex-bootstrap-info = [information]
channel-list = “C:” class-and-channels *(“,” class-and-channels)
class-and-channels = class “/” channel *(“,” channel)
class = 1*3DIGIT
channel = 1*3DIGIT
mac = “M:” 6hex-octet ; MAC address
hex-octet = 2HEXDIG
information = “I:” *(%x20-3A / %x3C-7E) ; semicolon not allowed
public-key = “K:” *PKCHAR ; DER of ASN.1 SubjectPublicKeyInfo encoded in “base64” as per [14]
PKCHAR = ALPHA / DIGIT / %x2b / %x2f / %x3d
```

The channel-list ABNF rule allows a list of IEEE 802.11 global operating class and channel (Annex E of [2]) value pairs to be specified. The MAC ABNF rule expresses the MAC address as a string of six hex-octets. The information ABNF rule allows freeform information to accompany the public key.

The bootstrapping information may be extended in future updates of the technical specification. Devices parsing this information should ignore unknown semicolon separated components in the dpp-qr and pkex-bootstrap-info instantiations to be forward compatible with such extensions.

5.3 Scanning a QR code

A QR code is a two-dimensional matrix barcode that encodes arbitrary strings of data. If a DPP bootstrapping key is (a component of) the data encoded, it is possible to obtain trust in such a key by scanning it as a QR code.

A public identity key represented as an ASN.1 SubjectPublicKeyInfo will be a binary string. By base64 encoding the binary string it is possible to obtain an alphanumeric character string that may be easily QR encoded. The length of the base64-encoded alphanumeric character string depends on the size of the public key. Using the compressed SubjectPublicKeyInfo format specified in section 4.1, it is possible to obtain the lengths listed in Table 4.

Table 4. Encoded length of ECC Public keys

Elliptic curve	base64 encoded length
P-256	80 characters
P-384	96 characters
P-521	120 characters

A version 4 QR code is able to encode 114 alphanumeric characters with a low error-correction rate. Version 5 is able to encode 154 alphanumeric characters and version 6 is able to encode 195 alphanumeric characters. Extra character capacity, in addition to the public key, is taken up by an identifying tag to indicate to a scanner application that it is reading a QR code for DPP, and for freeform additional ancillary information that may optionally be displayed or otherwise made use of.

The format of the data to QR encode a DPP bootstrapping information is given in section 5.2.1.

For example, a P-256 public key for a device that additionally indicates that it is operating on channel 1 and 36, can be represented as the following character string:

> DPP:C:81/1,115/36;K:MDkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDIgADM2206avxHJaHXgLMkq/24e0rsrfMP9K1Tm8gx+ovP0I=;;

which is 103 characters and can be easily encoded as a version 4 QR code. The QR code generated from this bootstrapping information is shown in Figure 6.

Figure 6. Example bootstrapping QR Code for P-256 Public Key with operating channels

Large amounts of ancillary information, public keys from elliptic curves with a larger prime field, or non-compressed SubjectPublicKey representations of a public key may require more QR code modules and therefore a higher version number. Larger version QR codes are denser and require a larger physical representation to ensure easy and consistent scanning. For example, a P-256 public key for a device that includes a MAC address of 01:02:03:04:05:06, and additionally indicates its serial number by “SN=4774LH2b4044, can be represented as the following character string:

DPP:I:SN=4774LH2b4044;M:010203040506;K:MDkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDIgADURzxmttZoIRIPWGoQMV00XHWCAQIhXruVWOz0NjlkIA=;;

which is 121 characters and can easily be encoded as a version 5 QR code. The bootstrapping QR code generated from this bootstrapping information is shown in Figure 7.

Figure 7. Example bootstrapping QR Code for P-256 public key with MAC address and serial number

A manufacturer of devices that enable QR code bootstrapping shall take into account the physical area available to affix a QR code as well as the size of public keys on the chosen elliptic curve and the amount of ancillary information to include in the QR code.

A public key received by scanning a QR code shall be validated before being accepted.

5.4 NFC

NOTE: THIS FEATURE HAS NOT BEEN TESTED IN THE DEVICE PROVISIONING PROTOCOL (DPP) CERTIFICATION PROGRAM.

5.4.1 Overview

NFC is a contactless technology designed for very short-range operation, approximately 10 cm or less. NFC is compliant with today’s field proven contactless smart card technology.

NFC may be used to convey a variety of different data structures, some as simple transfers, and others as more complex interactions referred to as NFC Protocols. The NFC Forum defines the Connection Handover protocol [24] to facilitate the "handover" from an NFC exchange to some other connection.

The main aspects that make NFC different and complementary to wireless network technology are:

1. Short distance: NFC is designed for a distance of approximately 10 cm or less to ensure that the intentional action of the user, bringing two NFC devices close to each other, is needed to trigger/initiate the communication.
2. Passive Communication via NFC Tags: NFC communications may occur between two NFC Devices that each are able to actively "write" (send) NFC data to, and "read" (receive) NFC data from, another NFC Device. Additionally, an NFC Tag is a type of unpowered passive NFC entity that is able to "write" (send) data once it is powered up by the induction provided by an active peer, which allows communication between a powered device and an unpowered device (one without a battery) such as a contactless smart card or a sticker.

NFC requires at least one of the NFC enabled entities to be portable.

5.4.2 NFC Connection Handover

NFC Connection Handover is an NFC protocol that enables NFC to convey connection information for devices and their services that have a presence on other non-NFC transports such as Wi-Fi, Bluetooth, and IP infrastructure. The NFC physical proximity interaction model helps to facilitate these other connections with a reduced need for additional user interaction.

NFC Connection Handover supports two interaction models: Static Handover and Negotiated Handover. In a Static Handover, an NFC Device reads an NFC Tag to acquire the parameters to connect to another (possibly non-NFC aware) device via the transports that the NFC Tag describes. In this interaction model, the NFC Tag assumes the role of "Handover Selector". In a Negotiated Handover, two NFC Devices negotiate who will be the "Handover Requestor", and then one assumes that role and the other takes the role of "Handover Selector".

5.4.2.1 DPP bootstrapping via Negotiated Handover

DPP Negotiated Handover Bootstrapping uses the NFC Connection Handover Negotiated Handover to allow two NFC Devices to directly exchange bootstrapping information. Negotiated Handover is built on top of the NFC Forum Logical Link Control Protocol (LLCP). If one of the devices has the intention to activate communications via a non-NFC communication technology, it may use the NFC Forum Connection Handover protocol [24] to announce possible communication means (potentially including configuration data) and request the other device to respond with a selection of matching technologies, possibly including necessary configuration data.

The negotiation portion of Negotiated Handover involves the two NFC Devices negotiating who will be the Handover Requestor and who will be the Handover Selector. There is no guarantee that the Initiator will be the Handover Requestor and the Responder will be the Handover Selector.

To use Negotiated Handover for DPP bootstrapping, the Initiator establishes an LLCP connection to the Responder and transmits its DPP Bootstrapping Information as a URI, as described in section 5.2, in an NFC Handover Request message. The Responder processes the handover request and responds with an NFC Handover Select message including its bootstrapping information. The Initiator and Responder shall perform mutual DPP Authentication when NFC dynamic handover is used.

Table 5 shows an example of a Handover Request message where the only available carrier is a DPP bootstrapping carrier. In practice, there may be multiple choices depending on the device capabilities. Also, the carrier power state may have one of the other values defined in the Connection Handover specification if the Wi-Fi radio is not yet active at the time of sending.

Table 5. NFC Handover Request message

Length	Value	Description
Handover Request Record		
1	0x91	Record Header: MB=1, ME=0, CF=0, SR=1, IL=0, TNF=WKT
1	0x02	Type Length : 2
1	0x0A	Payload Length: 10
2	0x48 0x72	Type: "Hr"
1	0x14	Version: 1.2
Alternative Carrier Record		
1	0xD1	Record Header: MB=1, ME=1, CF=0, SR=1, IL=0, TNF=WKT
1	0x02	Type Length: 2
1	0x04	Payload Length: 4
2	0x61 0x63	Type: "ac"
1	0x01	AC Header: Carrier Power State (Active)

Length	Value	Description
1	0x01	Carrier Data Reference Length: 1
1	0x31	Carrier Data Reference Value: "1"
1	0x00	Auxiliary Data Reference Count: 0
Wi-Fi DPP Bootstrapping Carrier Configuration Record		
1	0x5A	Record Header: MB=0, ME=1, CF=0, SR=1, IL=1, TNF=MIME
1	0x17	Type Length: 23
1	0x7A	Payload Length: 122
1	0x01	ID Length: 1
23	0x61 0x70 0x70 0x6c 0x69 0x63 0x61 0x74 0x69 0x6f 0x6e 0x2f 0x76 0x6e 0x64 0x2e 0x77 0x66 0x61 0x2e 0x64 0x70 0x70	Type: "application/vnd.wfa.dpp"
1	0x31	ID: "1"
1	0x00	URI Identifier Code: 0 (None - the "DPP" scheme isn't registered)
121	0x44 0x50 0x50 0x3a 0x49 0x3a 0x53 0x4e 0x3d 0x34 0x37 0x37 0x34 0x4c 0x48 0x32 0x62 0x34 0x30 0x34 0x34 0x3b 0x4d 0x3a 0x30 0x31 0x30 0x32 0x30 0x33 0x30 0x34 0x30 0x35 0x30 0x36 0x3b 0x4b 0x3a 0x4d 0x44 0x6b 0x77 0x45 0x77 0x59 0x48 0x4b 0x6f 0x5a 0x49 0x7a 0x6a 0x30 0x43 0x41 0x51 0x59 0x49 0x4b 0x6f 0x5a 0x49 0x7a 0x6a 0x30 0x44 0x41 0x51 0x63 0x44 0x49 0x67 0x41 0x44 0x55 0x52 0x7a 0x78 0x6d 0x74 0x74 0x5a 0x6f 0x49 0x52 0x49 0x50 0x57 0x47 0x6f 0x51 0x4d 0x56 0x30 0x30 0x58 0x48 0x57 0x43 0x41 0x51 0x49 0x68 0x58 0x72 0x75 0x56 0x57 0x4f 0x7a 0x30 0x4e 0x6a 0x6c 0x6b 0x49 0x41 0x3d 0x3b 0x3b	Bootstrapping URI as described in section 5.2

A Responder NFC device shall respond to a Handover Request message with a Handover Select message. Table 6 shows a Handover Select message, where the only available alternative carrier is DPP bootstrapping. In practice, there may be multiple alternative carrier choices from which to choose, and the Carrier Power State may have one of the other values defined in the Connection Handover specification if the Wi-Fi radio is not yet active at the time of sending. Once the Handover Select message has been received, the Initiator and Responder ought to have what they need to perform mutual DPP Authentication.

Table 6. NFC Handover Select Message

Length	Value	Description
Handover Select Record		
1	0x91	Record Header: MB=1, ME=0, CF=0, SR=1, IL=0, TNF=WKT
1	0x02	Type Length: 2
1	0x0A	Payload Length: 10
2	0x48 0x73	Type: "Hs"
1	0x14	Version: 1.2
Alternative Carrier Record		
1	0xD1	Record Header: MB=1, ME=1, CF=0, SR=1, IL=0, TNF=WKT
1	0x02	Type Length: 2
1	0x04	Payload Length: 4

Length	Value	Description
2	0x61 0x63	Type: "ac"
1	0x01	AC Header: Carrier Power State (Active)
1	0x01	Carrier Data Reference Length: 1
1	0x41	Carrier Data Reference Value: "A"
1	0x00	Auxiliary Data Reference Count: 0
Wi-Fi DPP Bootstrapping Carrier Configuration Record		
1	0x5A	Record Header: MB=0, ME=1, CF=0, SR=1, IL=1, TNF=MIME
1	0x17	Type Length: 23
1	0x7A	Payload Length: 122
1	0x01	ID Length: 1
23	0x61 0x70 0x70 0x6c 0x69 0x63 0x61 0x74 0x69 0x6f 0x6e 0x2f 0x76 0x6e 0x64 0x2e 0x77 0x66 0x61 0x2e 0x64 0x70 0x70	Type: "application/vnd.wfa.dpp"
1	0x41	ID: "A"
1	0x00	URI Identifier Code: 0 (None - the "DPP" scheme isn't registered)
121	0x44 0x50 0x50 0x3a 0x49 0x3a 0x53 0x4e 0x3d 0x34 0x37 0x37 0x34 0x4c 0x48 0x32 0x62 0x34 0x30 0x34 0x34 0x3b 0x4d 0x3a 0x30 0x31 0x30 0x32 0x30 0x33 0x30 0x34 0x30 0x35 0x30 0x36 0x3b 0x4b 0x3a 0x4d 0x44 0x6b 0x77 0x45 0x77 0x59 0x48 0x4b 0x6f 0x5a 0x49 0x7a 0x6a 0x30 0x43 0x41 0x51 0x59 0x49 0x4b 0x6f 0x5a 0x49 0x7a 0x6a 0x30 0x44 0x41 0x51 0x63 0x44 0x49 0x67 0x41 0x44 0x55 0x52 0x7a 0x78 0x6d 0x74 0x74 0x5a 0x6f 0x49 0x52 0x49 0x50 0x57 0x47 0x6f 0x51 0x4d 0x56 0x30 0x30 0x58 0x48 0x57 0x43 0x41 0x51 0x49 0x68 0x58 0x72 0x75 0x56 0x57 0x4f 0x7a 0x30 0x4e 0x6a 0x6c 0x6b 0x49 0x41 0x3d 0x3b 0x3b	Bootstrapping URI as described in section 5.2

5.4.2.2 DPP bootstrapping via Static Handover

DPP Static Handover Bootstrapping uses the NFC Connection Handover Static Handover. When the DPP Initiator is an NFC Device and the DPP Responder has or is represented by an NFC Tag, the NFC Tag contains the DPP bootstrapping information for the Responder. Touching the NFC Device to the NFC Tag, the DPP Initiator reads the DPP bootstrapping information for the DPP Responder.

Due to the unidirectional nature of the data transfer from an NFC Tag to an NFC Device in Connection Handover Static Handover, DPP does not support the scenario where the DPP Initiator is represented by an NFC Tag and the DPP Responder is an NFC Device.

The Handover Select message described in section 5.4.2.1 accurately describes a Handover Select message for either Static Handover or Negotiated Handover scenarios.

5.4.3 DPP bootstrapping via NFC URI record

If the additional infrastructure provided by the NFC Connection Handover is not necessary, DPP bootstrapping may be conveyed in an NFC Forum standard URI NDEF message containing a single URI record. Table 7 describes the structure of that message.

Table 7. DPP bootstrapping URI message

Length	Value	Description
Wi-Fi DPP Bootstrapping Carrier Configuration Record		
1	0xD1	Record Header: MB=1, ME=1, CF=0, SR=1, IL=0, TNF=WKT
1	0x01	Type Length: 1
1	0x7A	Payload Length: 122
1	0x55	Type: U
1	0x00	URI Identifier Code: 0 (None - the "DPP" scheme isn't registered)
121	0x44 0x50 0x50 0x3a 0x49 0x3a 0x53 0x4e 0x3d 0x34 0x37 0x37 0x34 0x4c 0x48 0x32 0x62 0x34 0x30 0x34 0x34 0x3b 0x4d 0x3a 0x30 0x31 0x30 0x32 0x30 0x33 0x30 0x34 0x30 0x35 0x30 0x36 0x3a 0x4b 0x3a 0x4d 0x44 0x6b 0x77 0x45 0x77 0x59 0x48 0x4b 0x6f 0x5a 0x49 0x7a 0x6a 0x30 0x43 0x41 0x51 0x59 0x49 0x4b 0x6f 0x5a 0x49 0x7a 0x6a 0x30 0x44 0x41 0x51 0x63 0x44 0x49 0x67 0x41 0x44 0x55 0x52 0x7a 0x78 0x6d 0x74 0x74 0x5a 0x6f 0x49 0x52 0x49 0x50 0x57 0x47 0x6f 0x51 0x4d 0x56 0x30 0x30 0x58 0x48 0x57 0x43 0x41 0x51 0x49 0x68 0x58 0x72 0x75 0x56 0x57 0x4f 0x7a 0x30 0x4e 0x6a 0x6c 0x6b 0x49 0x41 0x3d 0x3b 0x3b	Bootstrapping URI as described in section 5.2

5.5 Bluetooth

NOTE: THIS FEATURE HAS NOT BEEN TESTED IN THE DEVICE PROVISIONING PROTOCOL (DPP) CERTIFICATION PROGRAM.

5.5.1 Overview

Bluetooth Low Energy (BLE) provides the ability to advertise or scan for discovery of peer device information within a shorter communication range than Wi-Fi. Prior to Bluetooth 5.0, the advertising/scanning payload was limited to 29 bytes. Bluetooth 5.0 [36] allows an advertising/scanning payload of up to 255 bytes.

A Responder may advertise DPP bootstrapping information using BLE. An Initiator may discover and obtain DPP bootstrapping information using BLE. The Responder enters bootstrapping mode and begins to advertise the DPP bootstrapping URI on an auxiliary channel. Figure 8 gives an example message flow that describes the interaction between the Initiator and the Responder for BLE Bootstrapping when the Configurator is acting as the Initiator.

The Enrollee enters bootstrapping mode and advertises the secondary channel information associated with the bootstrapping information advertisements. These advertisements are periodically transmitted on primary BLE channels using ADV_EXT_IND packets. The advertisements on the primary channels include secondary channel information with respect to the next transmission on the secondary channel. The Enrollee also advertises the DPP bootstrapping information on the secondary channel using ADV_AUX_IND packets. The Enrollee alternates transmissions on the primary and secondary channels as shown in Figure 8. In addition, the Enrollee sends AUX_ADV_RSP which includes the Enrollee's bootstrapping information as shown in Figure 9, if the Enrollee receives AUX_ADV_REQ from the Configurator.

The Configurator enters bootstrapping mode and listens on the primary BLE channels until it discovers the Enrollee's advertisement. The Enrollees around the Configurator sends ADV_EXT_IND to advertise that they support the bootstrapping over BLE. The Configurator obtains the secondary channel information and then listens for the advertisement on the secondary channel, which includes the DPP Bootstrapping Information. The Configurator should display the list of the discovered Enrollees. If the user selects one of the Enrollees as the target, the Configurator does the following.

- If the Configurator has received an AUX_ADV_IND packet from the Enrollee, the BLE bootstrapping is completed.
- If the Initiator does not receive any AUX_ADV_IND from the Enrollee, the Initiator sends an AUX_SCAN_REQ packet to obtain the bootstrapping information from the Enrollee as shown in Figure 9.

When the Configurator receives AUX_SCAN_RSP including the bootstrapping information, the BLE bootstrapping is completed.

Once the bootstrapping is completed, the DPP Authentication and Configuration over Wi-Fi process follow.

The exchange of the bootstrapping information from the Enrollee to the Configurator shall be performed at close proximity. For this purpose, the transmission power level for DPP bootstrapping shall be lowered.

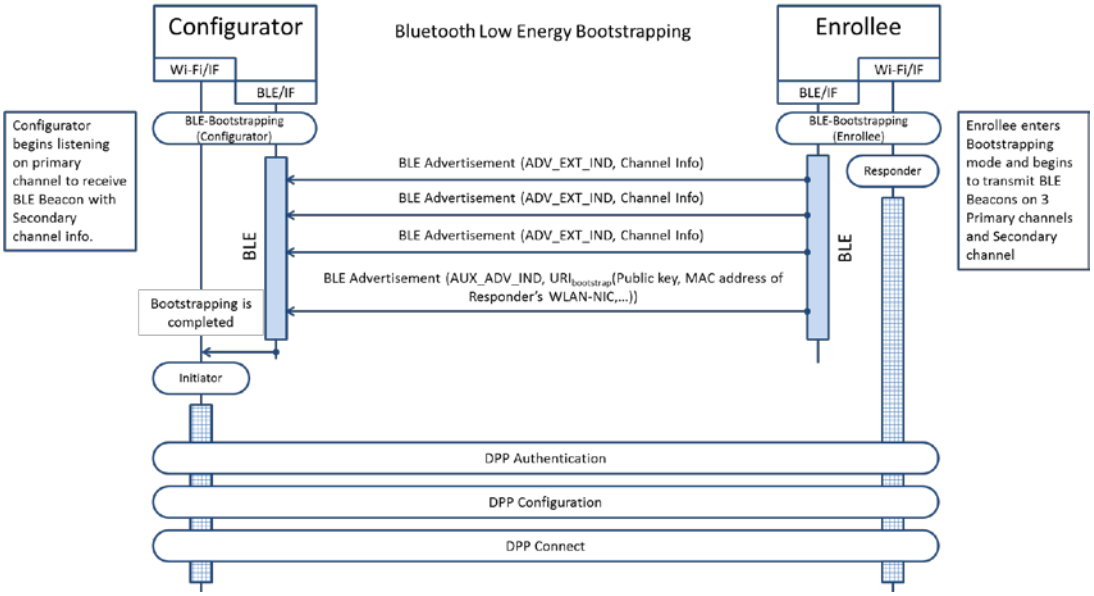

Figure 8. Example bootstrapping message sequence using BLE passive scan

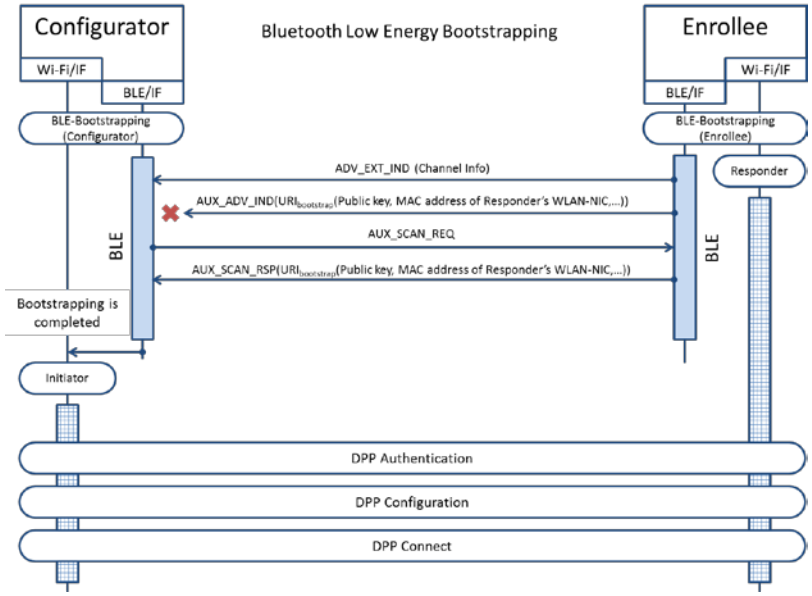

Figure 9. Example bootstrapping message sequence using BLE active scan

If mutual authentication is required for bootstrapping, the Initiator and Responder exchange each other's bootstrapping information.

5.5.2 Responder procedures

The Responder is triggered to enter bootstrapping mode. The trigger conditions for bootstrapping mode are beyond the scope of this specification. The Responder configures the BLE advertisement on the primary channel according to [36], with advertising information shown in Table 8.

Table 8. Primary channel advertisement parameters

Parameter	Value
Advertisement type	ADV_EXT_IND
Mode	Non-Connectable and Non-Scannable Undirected with auxiliary packet
ADI	Set to 0
AuxPtr	Contains the Secondary Channel information
TxPower	The TxPower as specified in [36]
Payload:	
UUID	Wi-Fi Alliance 16bit UUID for Bluetooth <to be assigned>
Type	0x01 - DPP bootstrapping
Length	0
Data	0

The Responder shall advertise the Bootstrapping URL on a secondary channel indicated in the primary channel advertisement. As shown in Figure 8, the bootstrapping URL shall include the MAC address of the Responder's Wi-Fi interface for the Responder and Initiator complete DPP Authentication. The secondary advertisement information is given in Table 9.

Table 9. Secondary channel advertisement parameters

Parameter	Value
Advertisement type	ADV_AUX_IND
ADI	Set to 0
TxPower	The TxPower as specified in [36]
Payload:	
UUID	Wi-Fi Alliance 16bit UUID for Bluetooth <to be assigned>
Type	0x02 (DPP URI)
Length	URI length
URI Data	URI encoded as per [37]

The Responder shall advertise the Bootstrapping information for 10 seconds, alternating advertisements on the primary and secondary channels. The period between advertisement transmissions is implementation specific.

5.5.3 Initiator procedures

The Initiator is triggered to enter BLE bootstrapping mode. The trigger conditions are implementation dependent. Once triggered, the Initiator begins listening on primary channels for a Responder advertisement.

The Initiator searches for extended advertisements on the primary channel that contain the DPP UUID in their payload. Receiving an extended advertisement on the secondary channel from the Responder, the Initiator parses the

bootstrapping information. If the Initiator obtains the bootstrapping information, it enables its own Wi-Fi interface if necessary. Then, it takes a role as an Initiator which is ready to initiate DPP Authentication based on the bootstrapping information.

5.6 PKEX: Proof of knowledge of a shared code, key, phrase, or word

Yet another technique is to mask bootstrapping information with a shared code/key/phrase/word (hereinafter, “code”) and rely on knowledge of the shared code to unmask the bootstrapping key. If a peer is able to prove it knows and can use the shared code, the peer’s bootstrapping key can be trusted. Additionally, optional bootstrapping information that is associated with the exchanged key may be sent in this protocol and are hidden from attackers. The only information leaked to an attacker by an active attack is whether a single guess of the code is correct or not. The larger the pool from which the code was drawn, the lower the probability that that guess will be correct. The trust placed in the public key received in this protocol is inversely proportional to the probability of an attacker guessing the code outright.

If both sides have a user interface, this technique can be used to bootstrap trust by exchanging bootstrapping information including the bootstrapping keys that are be used for a DPP exchange requiring mutual authentication. This bootstrapping technique should never be run multiple times with the same code to different peers.

The PKEX protocol [23] is used to establish trust in a peer’s bootstrapping key, and correspondingly allow the peer to establish trust in the device’s bootstrapping key using a shared code. Optionally, a non-secret identifier for the code can be transmitted to support the case where a PKEX implementation may be provisioned to connect to a plurality of devices and needs to know which code to use to process a received PKEX frame. If an optional code identifier is used, it shall be a UTF-8 string not greater than eighty (80) octets that is provisioned at the same time as the shared code.

5.6.1 PKEX preliminaries

The PKEX protocol requires an identity, a public key, and an identifier for the finite cyclic group to which the public key belongs. The PKEX identities are the MAC addresses of the peers. The *subjectPublicKey* from the device’s bootstrapping key is used as the public key and the particular elliptic curve identified by the AlgorithmIdentifier parameters in the device’s bootstrapping key is converted to a finite cyclic group using the registry maintained by IANA as “Group Description” attributes for IETF RFC 2409 [39], [40]. PKEX uses compact output, per RFC 6090 [5], by denoting certain ECDH secret output with the x-coordinate only, per the notation from section 3.2.4. PKEX does not use RFC 6090 compact representation, all transmitted public keys are sent using both the x- and y-coordinates per section 3.2.4.

Due to the nature of the PKEX exchange, the public keys exchanged by each device shall be from the same finite cyclic group.

In lieu of specific channel information obtained in a manner outside the scope of this specification, PKEX responders shall select one of the following channels:

- 2.4 GHz: Channel 6 (2.437 GHz)
- 5 GHz: Channel 44 (5.220 GHz) if local regulations permit operation only in the 5.150 – 5.250 GHz band and Channel 149 (5.745 GHz) otherwise
- 60 GHz: Channel 2 (60.48 GHz)

The Initiator shall iterate through the list of channels on the supported bands. If the Initiator of PKEX has learned the MAC address of the Responder in a manner outside the scope of this specification, it shall send its PKEX Exchange request to that MAC address, otherwise it shall send its PKEX Exchange Request to the broadcast MAC address.

NOTE: This may require the use of a code identifier to allow the responder to identify the specific code.

The protocol consists of two phases: an exchange phase and a commitment-reveal phase. In the exchange phase, the peers exchange ephemeral keys encrypted with the code. In the commit-reveal phase, the peers cryptographically bind their bootstrap private key to exchange and reveal their public key.

The protocol requires fixed elements in the group from which the public keys were derived. These fixed elements are role-specific—one for the Initiator, one for the Responder—and need not be secret. Elements for supported groups to use with PKEX are listed in Appendix B. The generator, G, used in PKEX is taken from the definition of the group’s domain parameter set.

Descriptively, the bootstrapping key that the Initiator wishes to share with the Responder is A with a private analog of a, and the bootstrapping key the Responder wishes to share with the Initiator is B with a private analog of b.

PKEX maintains a counter, t, of unsuccessful authentications for a given code. When a code and optionally a code identifier is configured, the counter is set to zero. Each unsuccessful authentication increments the counter by one. When the counter reaches five, the code, and the code identifier if provisioned, is irretrievably deleted.

The code, and code identifier if provisioned, should be deleted upon successful completion of PKEX; codes should not be reused.

The hash algorithm used with PKEX shall be based on the elliptic curve used and determined by Table 3 in section 3.2.3.

The AAD used with PKEX shall consist of two components in the following order: (1) the DPP header, as defined in Table 19, from the OUI (inclusive) to the DPP frame type (inclusive); and (2) a single octet of the value zero or one depending on whether the sender is the initiator or responder, respectively.

DPP Public Action frames from section Figure 20 are used to form PKEX messages.

Test vectors for an example run of PKEX using NIST p256 are given in Appendix C.

5.6.2 PKEX exchange phase

The Initiator selects a group from which it has obtained a public/private key pair it wishes to bootstrap with a peer. Once the group and keypair have been identified, the Initiator hashes its identity with the code, optionally including the code identifier, and multiplies the result with the Initiator-element for the selected group, denoted Pi, to generate an encrypting element, Q_i. If Q_i is the point-at-infinity, the code shall be deleted and the user should be notified to provision a new code. The Initiator then generates a random ephemeral keypair, x/X, in the same group, adds X to Q_i, thereby encrypting it, and sends the selected group, and optionally the code identifier, and the result, denoted M, to the Responder as a PKEX Exchange Request:

Q_i = H(MAC-Initiator | [identifier |] code) * P_i

X = x * G

M = X + Q_i

Initiator → Responder: group, [identifier,] M

Where [identifier |] is only in brackets is optionally included when a code identifier is used. If the Initiator does not receive a response to its PKEX Exchange Request within 200 ms it should retransmit the PKEX Exchange Request a maximum of five times before attempting to change channels. If the PKEX Initiator has exhausted all possible channels and has not received a response it shall abandon PKEX.

The Responder receives the PKEX Exchange Request and checks whether the selected finite cyclic group is acceptable. If it is not acceptable, the Responder generates a PKEX Exchange Response with the DPP Status set to STATUS_BAD_GROUP and offers an acceptable alternative in the finite cyclic group field:

Responder → Initiator: DPP Status, group

Otherwise, the Responder obtains M and validates M per section 3.2.4. If M is not valid, the counter t is incremented and the exchange silently fails. Otherwise, the Responder computes its own version of Q_i in the indicated group and decrypts M.

Qi = H(MAC-Initiator | [identifier |] code) * P_i

X’ = M - Q_i

The resulting ephemeral key, denoted X’, is checked whether it is the point-at-infinity. If it is not valid, the protocol silently fails and the counter t is incremented. Otherwise, the Responder hashes his identity with the code, and optionally the code identifier, and multiplies the result with the Responder-element, denoted P_r, to generate an encrypting element, Q_r. The Responder then generates a random ephemeral keypair, y/Y, encrypts Y with Q_r to obtain the result, denoted N. The responder then constructs a PKEX Exchange Response with a DPP Status, and optionally the code identifier if the Initiator included it in the PKEX Exchange Request, and N.

If Q_r is the point-at-infinity the DPP Status is set to STATUS_BAD_CODE and the following PKEX Response is sent to the Initiator.

Responder → Initiator: DPP Status, [identifier]

If the Responder set the status to STATUS_BAD_CODE, the code shall be irretrievably deleted and the protocol shall unsuccessfully fail. The user should be notified to provision a new code.

If Q_r is not the point-at-infinity, the DPP Status is set to STATUS_OK and the PKEX Exchange Response is sent to the Initiator:

Q_r = H(MAC-Responder | | [identifier |] code) * P_r

Y = y * G

N = Y + Q_r

Responder → Initiator: DPP Status, [identifier,] N

At this time, the Responder computes secret element K by multiplying the Initiator's ephemeral key X' by the Responder's ephemeral private key, and generates a shared secret, z, for AES-SIV whose length is determined by Table 3:

K = y * X'

z = HKDF(<>, MAC-Initiator | MAC-Responder | M.x | N.x | code, K.x)

The Responder shall delete the secret element K upon generation of z.

The Initiator receives the PKEX Exchange Response and checks DPP Status. If it is STATUS_BAD_GROUP, the Initiator inspects the offered group. If the group offers less security than the offered group, then the Initiator should ignore this message and attempt its original PKEX Exchange Request due to the unsecured nature of this response. If the group offers equivalent or better security than the offered group, then the Initiator may choose to abort its original request and try another exchange using the group offered by the Responder. If DPP Status is STATUS_BAD_CODE the Initiator should verify that Q_r gets computed to be the point-at-infinity and if so it shall irretrievably delete the code and fail the exchange. The user should be notified to provision a new code in this case. If Q_r is not the point-at-infinity or if DPP status is any value other than STATUS_OK, the Initiator ignores the message. Otherwise, DPP Status is STATUS_OK and the group is acceptable. Therefore, the Initiator obtains N, validates N per section 3.2.4. If N is not valid, the Initiator increments the counter t and silently fails the exchange. Otherwise, it computes its own version of Q_r and decrypts N.

Q_r = H(MAC-Responder | [identifier |] code) * P_r

Y' = N - Q_r

Where [identifier |] is only included when a code identifier is used.

The resulting ephemeral key, denoted Y', is then checked whether it is the point-at-infinity. If it is not valid the protocol ends unsuccessfully and the counter t is incremented. Otherwise, the Initiator multiplies the Responder's ephemeral key, Y', by the Initiator's ephemeral private key to produce another secret element, K, and the shared secret key, z,

K = x * Y'

z = HKDF(<>, MAC-Initiator | MAC-Responder | M.x | N.x | code, K.x)

The protocol advances to the next stage.

5.6.3 PKEX commit-reveal phase

The Initiator then multiplies the Responder's ephemeral key, Y', by the Initiator's private bootstrapping key, a, generates a shared key, J, and signs a commitment consisting of its MAC address, its public bootstrapping key, and the two ephemeral keys:

J = a * Y'

u = HMAC(J.x, MAC-Initiator | A.x | Y'.x | X.x)

The Initiator encrypts and authenticates this commitment, its bootstrapping key, and any optional additional information in the ABNF format from section 5.2.1, denoted “bootstrapping info”, for the Responder using AES-SIV.

The Initiator sends the encrypted bootstrapping data to the Responder as a PKEX Commit-Reveal Request:

Initiator → Responder: $\{A, u, [\text{bootstrapping info}]\}_z$

Upon receipt of this message, the Responder decrypts the Initiator’s bootstrapping data using secret z.

If decryption of the message fails, the protocol fails, the counter t is incremented, and an alert should be given to the user. Otherwise, the decrypted bootstrapping key, A’, is checked for validity. If the decrypted bootstrapping key is not valid, the protocol fails. Otherwise, it continues and the Responder then computes the shared key and verifies the Initiator’s commitment:

J’ = y * A’

u’ = HMAC(J’.x, MAC-Initiator | A’.x | Y.x | X’.x)

If u’ does not equal u, then the protocol fails and the counter t is incremented. Otherwise, the Responder continues. At this point the Responder has authenticated the Initiator and trusts the Initiator’s bootstrapping key A.

The Responder then multiplies the Initiator’s ephemeral key X’ by the Responder’s private bootstrapping key, b, to produce a shared key, L, and signs a commitment consisting of his MAC address, his public bootstrapping key, and the two ephemeral keys:

L = b * X’

v = HMAC(L.x, MAC-Responder | B.x | X’.x |Y.x)

Next the Responder uses z, to encrypt its bootstrapping key, its confirmation, and any optional additional information in the ABNF format from section 5.2.1, denoted “bootstrapping info”, for the Initiator with AES-SIV. The Responder sends the encrypted bootstrapping data to the Initiator as a PKEX Commit-Reveal Response:

Responder → Initiator: $\{B, v, [\text{bootstrapping info}]\}_z$

Upon receipt of this message, the Initiator decrypts the Responder’s bootstrapping information. If decryption fails, the protocol fails, the counter t is incremented, and an alert should be given to the user. Otherwise, the decrypted bootstrapping key, B’, is checked for validity. If the decrypted bootstrapping key is not valid, the protocol fails. Otherwise, it continues and the Initiator computes the shared key and verifies the Responder’s commitment:

L’ = x * B’

v’ = HMAC(L.x, MAC-Responder | B’.x | X.x | Y’.x)

If v’ does not equal v then the protocol fails and the counter t is incremented. Otherwise, it succeeds and the Initiator trusts the Responder’s bootstrapping key B.

Upon successful completion of the PKEX protocol, the device has the peer’s trusted public key, and the peer has the device’s trusted public key. The code, and code identifier if provisioned, should be deleted at this point. The peer’s trusted public key, and the finite cyclic group to which it belongs, is used to construct a SubjectPublicKeyInfo representation of the peer’s bootstrapping key per section 4.1: the AlgorithmIdentifier algorithm is the object ecPublicKey, the finite cyclic group is converted back to the AlgorithmIdentifier parameters defining the elliptic curve, and the public key is compressed to become the subjectPublicKey.

The device that initiated the PKEX protocol shall initiate the DPP Authentication protocol immediately after successful completion of the PKEX protocol. The same MAC addresses and the same channel shall be used for both protocols.

6 DPP authentication

6.1 Overview

The goal of DPP is an authenticated key, a PMK and a PMKSA. The security of DPP ensures that the PMK is strictly pairwise and only two parties know it. The PMK and PMKSA are to be used by a device to connect to another device as part of a network access protocol (see section 2).

It is assumed that the Initiator has a way to bootstrap trust in the Responder's public bootstrapping key, using, for instance, one of the methods from section 5. Optionally, the Initiator can also provide its public bootstrapping key to the Responder using one of the methods in section 5 to enable mutual authentication.

Once the Initiator has obtained the Responder's representation of its public bootstrapping key, the enrollment process is initiated. The Initiator now actively searches for the device to configure, or for devices to perform configuration (depending on whether it is assuming the role of configurator or enrollee, respectively) by beginning the DPP enrollment protocol.

Unmanaged devices that are not capable of acting as the Initiators (such as headless devices) are quiescent, they listen on a channel or channels to be discovered, enrolled, and configured by a Configurator acting as the Initiator. They process all DPP authentication frames received and respond to those directed at them, dropping all others.

During the initial exchange of DPP authentication frames, the Initiator and Responder agree upon the roles they will play—Configurator or Enrollee—during provisioning. Regardless of Initiator or Responder role, the Configurator's protocol key is always ephemeral (used once and thrown away) and the Enrollee's protocol key always becomes its network access provisioning key.

6.2 DPP Authentication protocol

The DPP Authentication protocol uses trusted public bootstrapping keys, obtained using a technique from section 5, to strongly authenticate the Responder to the Initiator and optionally the Initiator to the Responder. It consists of a 3-message exchange, including the exchange of ephemeral "protocol keys", and generates a shared secret and authenticated key. Ephemeral protocol keys ensure that generated secrets are distinct even if authenticating bootstrapping keys are reused in a subsequent run of the DPP Authentication protocol. The protocol key of the Enrollee is used as Network Access key later in the DPP Configuration and DPP Introduction protocol. The protocol key of the Configurator is discarded after the DPP Authentication protocol.

The Responder listens on channels included in the channel list provided in the bootstrapping information or on any available channel if the optional channel list is not provided, waiting to receive a DPP Authentication Request frame. Upon successful receipt of a DPP Authentication Request frame, the Responder transmits a DPP Authentication Response frame to the Initiator.

The Initiator shall determine the list of possible DPP authentication channels by taking the intersection of the channels it supports under the current regulatory requirements and the channels that are present in the bootstrapping information, if included. If the bootstrapping information does not include the optional channel list, the Initiator uses all the channels it supports under the current regulatory requirements as the list of possible DPP authentication channels. If the list of possible DPP authentication channels is empty, DPP authentication cannot be performed and the Initiator should notify the user.

The Initiator shall select a channel from the list of possible DPP authentication channels and transmit a DPP Authentication Request frame. If the bootstrapping information includes the MAC address of the Responder, this frame shall be sent as a unicast frame to that address; otherwise, this frame is sent to the broadcast address. The Initiator sets a timer to ten seconds and listens to receive a DPP Authentication Response frame from the Responder. When using unicast DPP Authentication Request frame, the Initiator may leave the channel if no ACK frame is received for the DPP Authentication Request frame (for example, to allow the radio to be used on other channels for concurrent operations) or if the ACK frame is received but no DPP Authentication Response frame is received within ten seconds. If the Responder responds with a DPP Authentication Response frame, the timer is cleared and discovery completes. If the timer expires without a response from the Responder, and an ACK frame was received for the DPP Authentication Request frame, the Initiator may move to the next available channel without retransmitting the DPP Authentication Request frame. Otherwise, if the timer expires without a response from the Responder, the Initiator rebroadcasts the DPP Authentication Request frame and resets the timer. This process shall be retried no less than five times. If the Responder never responds,

discovery is abandoned and the Initiator attempts DPP Authentication on another channel in the list of possible DPP authentication channels, if more than one channel is available. This iteration over the possible channels continues until a valid response is received or all the possible channels have been attempted at least once.

Upon successful validation of the DPP Authentication Response frame, the Initiator shall transmit a DPP Authentication Confirm frame to the Responder to complete the authentication.

Once DPP Authentication is complete, the peer acting as the enrollee transmits a DPP Configuration Request frame to the Configurator.

Upon successful receipt of the DPP Configuration Request frame, the Configurator transmits the DPP Configuration Response frame to complete provisioning of the Enrollee.

Keys are identified by a “P” or “B” for protocol or bootstrapping respectively. Private keys are lowercase and public keys are uppercase. Subscripts “I”, and “R” represent the participant roles, Initiator and Responder, respectively. For instance, B_I is the Initiator’s public bootstrapping key and b_I is the Initiator’s private bootstrapping key. Table 10 below lists the complete notation set. Protocol keys are ephemeral and shall never be reused in DPP Authentication exchanges; bootstrapping keys can be static and reused.

Table 10. DPP authentication key notation

Responder (R) \ Initiator (I)	Bootstrapping key (B,b)	Protocol key (P,p)
Public key (Upper case)	B_I, B_R	P_I, P_R
Private key (Lower case)	b_I, b_R	p_I, p_R

The security of the DPP Authentication protocol requires each entity to prove possession of private keys using AES-SIV (RFC 5297 [3]). The private keys are used to generate secrets through a Diffie-Hellman exchange and derivatives of those secrets are used to protect messages sent in the DPP Authentication protocol. The ability of an entity to wrap messages with AES-SIV using these secrets proves possession of private keys used to generate the secrets.

All messages used in the DPP Authentication protocol are 802.11 Public Action frames (see section Figure 20) containing a DPP-specific header and a series of attributes (see section 8.1). Attributes wrapped by AES-SIV are represented in DPP protocol messages as a Wrapped Data attribute. The value of the Wrapped Data attribute shall be the ciphertext, including the authenticating tag, output by AES-SIV. Unwrapping of AES-SIV-protected data results either in one or more attributes, or failure.

Invocations of AES-SIV in the DPP Authentication protocol that produce ciphertext that is part of an additional AES-SIV invocation do not use AAD, in other words, the number of AAD components is set to zero. All other invocations of AES-SIV in the DPP Authentication protocol shall pass a vector of AAD having two components of AAD in the following order: (1) the DPP header, as defined in Table 19, from the OUI (inclusive) to the DPP frame type (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 Wrapped Data attribute shall be the last attribute in a DPP Public Action frame.

6.2.1 DPP capabilities negotiation

When initiating the DPP Authentication protocol, the Initiator acts either as the Configurator or the Enrollee, or it indicates that it can be either. The Initiator and the Responder exchange Capabilities information during DPP Authentication to establish that one device acts as the Configurator and the other acts as the Enrollee.

The Initiator and Responder include their capabilities in the DPP Authentication Request and DPP Authentication Response frames, respectively. The summary of capabilities values and fields is given in Table 11.

Table 11. DPP authentication capabilities role summary

Initiator	Responder	DPP Status
Configurator	Configurator	STATUS_NOT_COMPATIBLE
Configurator	Enrollee	STATUS_OK
Enrollee	Configurator	STATUS_OK
Enrollee	Enrollee	STATUS_NOT_COMPATIBLE
Configurator or Enrollee	Configurator	STATUS_OK
Configurator or Enrollee	Enrollee	STATUS_OK

The Initiator shall include its role(s) in the DPP Authentication Request frame. After successfully receiving a DPP Authentication Request frame, the Responder shall compare the value of the Role field from the Initiator to the expected role and respond with a DPP Authentication Response frame with a DPP Status code set to a DPP Status indicated in Table 11. The Responder shall include its role in the Capabilities field as well as a DPP Status field in the DPP Authentication Response frame. The DPP Status field shall include a status corresponding to the DPP Status column of Table 11 with a value in Table 36. After successfully receiving the DPP Authentication Response frame, the Initiator shall compare the value of the Role field from the Responder to the expected role and respond with a DPP Authentication Confirm frame with a DPP Status code set to a result indicated in Table 11.

6.2.2 DPP authentication request

The Initiator generates a random nonce whose length is determined according to Table 3, generates a protocol keypair in the agreed upon domain parameter set (determined by the Responder's bootstrapping key), performs a Diffie-Hellman to generate a shared secret M and a first intermediate key, k1. The Initiator wraps its nonce and its capabilities in the first intermediate key using AES-SIV. It then performs a SHA256 hash on the DER encoded ASN.1 SubjectPublicKeyInfo of the Responder's public bootstrapping key, performs a SHA256 hash on the DER encoded ASN.1 SubjectPublicKeyInfo of its public bootstrapping key. The Initiator then places the hash of the Responder's public bootstrapping key, the hash of its public bootstrapping key, an Initiator Protocol Key attribute indicating its public protocol key (PI), a Wrapped Data attribute that contains the Initiator Nonce attribute and the Initiator Capabilities attribute in a DPP Authentication Request frame. If the Initiator prefers to use a different channel for going through the rest of the DPP Authentication and DPP Configuration exchanges to avoid off channel operations (for example, when operating as an AP), the Initiator adds the optional Channel attribute to the message.

$M = p_I * B_R$

k1 = HKDF(<>, "first intermediate key", M.x)

Initiator → Responder: SHA256(B_R), SHA256(B_I), PI, [Channel,] {I-nonce, I-capabilities}$_{k1}$

6.2.3 DPP authentication response

The Responder receives the DPP Authentication Request frame and checks whether a SHA256 hash of the DER encoded ASN.1 SubjectPublicKeyInfo of its public bootstrapping key, B_R, is in the message. If not, it discards the message and returns to its quiescent state. If a hash of its key is in the message, it next checks whether it has a copy of the Initiator's public bootstrapping key, B_I (whose SHA256 hash matches that in the message). If so, the Responder may perform mutual authentication. Specifically, the Responder shall request mutual authentication when the hash of Responder bootstrapping key in the authentication request indexes an entry in the bootstrapping table corresponding to a bidirectional bootstrapping method, for example, PKEX or BTLE.

If the optional Channel attribute is included, the Responder determines whether it can use the requested channel for the following exchanges. If so, it sends the DPP Authentication Response frame on that channel. If not, it discards the DPP Authentication Request frame without replying to it.

The Responder generates the shared secret M and intermediate key k1 and attempts to unwrap the Initiator's nonce using AES-SIV.

$M = b_R * P_I$

k1 = HKDF(<>, "first intermediate key", M.x)

If AES-SIV returns FAIL, the Responder abandons the exchange. Otherwise, it checks the Initiator's Capabilities If the Responder is not capable of supporting the role indicated by the Initiator, it shall respond with a DPP Authentication Response frame indicating failure by adding the DPP Status field set to STATUS_NOT_COMPATIBLE, a hash of its public bootstrapping key, a hash of the Initiator's public bootstrapping key if it is doing mutual authentication, and Wrapped Data element consisting of the Initiator's nonce and the Responder's desired capabilities wrapped with k1:

Responder → Initiator: DPP Status, SHA256(B_R), [SHA256(B_I),], { I-nonce, R-capabilities}$_{k1}$

The Responder then aborts the exchange.

If the Responder needs more time to respond due as to complete bootstrapping of the Initiator's bootstrapping key, it shall respond with a DPP Authentication Response frame indicating that it will reply in full at a later time by adding the DPP Status field set to STATUS_RESPONSE_PENDING, hashes of both bootstrapping keys obtained from the DPP Authentication Request frame, and wrapped data consisting of the Initiator's nonce and the Responder's capabilities wrapped with k1.

Responder → Initiator: DPP Status, SHA256(B_R), SHA256(B_I), { I-nonce, R-capabilities}$_{k1}$

The Responder shall not abort the exchange. When the Initiator's bootstrapping key has been bootstrapped, the Responder shall continue by sending a full DPP Authentication Response frame as follows.

1. The Responder first selects Capabilities that support the Initiator—for example, if the Initiator states it is a Configurator, then the Responder takes on the Enrollee role.
2. Next, the Responder generates its own nonce, R-nonce, whose length is determined according to Table 3, a protocol key pair (PR/pR), one or two intermediate elements depending on whether mutual authentication is performed, a shared secret, L, and encryption keys, k2 and ke:

   $N = p_R * P_I$

   k2 = HKDF(<>, "second intermediate key", N.x)

   [$L = ((b_R + p_R) \text{ modulo } q) * B_I$]

   ke = HKDF(I-nonce | R-nonce, "DPP Key", M.x | N.x [| L.x])

   Where q is the order of the elliptic curve group and where L is only computed and used in the computation of ke when doing mutual authentication.
3. The Responder then generates an authenticating tag:

   R-auth = H(I-nonce | R-nonce | P_I.x | P_R.x | [B_I.x |] B_R.x | 0)

   Where 0 is a single octet with the value zero and where [B_I.x |] is only used in the computation of R-auth when doing mutual authentication.
4. The Responder then wraps the Responder Authenticating Tag attribute as shown in step 3 with the encryption key ke and then wraps the Initiator Nonce, Responder Nonce, Responder Capabilities attributes and the wrapped Responder Authenticating Tag attribute with the intermediate key, k2.
5. The Responder then places a DPP Status of STATUS_OK, a hash of its public bootstrapping key, a hash of the Initiator's public bootstrapping key if it is doing mutual authentication, a public Responder Protocol Key attribute, and a Wrapped Data attribute as contructed in step 4 in a DPP Authentication Response frame.
6. The DPP Authentication Response frame is then transmitted to the Initiator.

   Responder → Initiator: DPP Status, SHA256(B_R), [SHA256(B_I),] PR, { R-nonce, I-nonce, R-capabilities, { R-auth }ke }$_{k2}$

where [SHA256(B_I),] is only present when doing mutual authentication.

6.2.4 DPP authentication confirm

Upon receipt of a DPP Authentication Response frame, the Initiator checks that the DPP Status is set to STATUS_OK. If it is not, the Initiator unwraps the wrapped data portion of the frame using k1, and checks the Responder's indicated capabilities. If unwrapping fails, the Initiator aborts the exchange. If unwrapping is successful and DPP Status is set to STATUS_NOT_COMPATIBLE, the Initiator aborts the exchange and may initiate back to the Responder with a different set of Initiator capabilities. If unwrapping is successful and DPP Status is set to STATUS_RESPONSE_PENDING, the Initiator shall not abort the exchange and shall wait for a full DPP Authentication Response frame. The Initiator is advised to set a timer to clean up the nascent connection if a response is not received in an acceptable amount of time. The time limit is not specified in this specification.

If DPP Status is set to STATUS_OK, the Initiator checks whether a hash of its public bootstrapping key is included in the response. If so, the Initiator performs mutual authentication, otherwise it does not.

The Initiator then validates the received public protocol key, P_R. If it is not valid, the protocol terminates, otherwise the Initiator uses the received public protocol key to generate the intermediate key, k2:

$N = pI * P_R$

k2 = HKDF(<>, "second intermediate key", N.x)

It then unwraps the nonces and the Responder's capabilities key, and the wrapped Responder's authentication tag using k2. If unwrapping returns FAIL, it aborts the exchange. Otherwise it verifies that the received I-nonce is the same as the I-nonce sent in the DPP Authentication Request frame. If they differ, the Initiator aborts the exchange. Otherwise, it verifies that the Responder's capabilities are compatible with its own (that the Responder has not chosen the same non peer-to-peer role that it chose). If they are not, the Initiator shall respond with a DPP Authentication Confirm frame indicating failure by adding the DPP Status field set to STATUS_NOT_COMPATIBLE, a hash of the Responder's public bootstrapping key, a hash of its public bootstrapping key if it is doing mutual authentication, and wrapped data consisting of the Responder's nonce wrapped with k2,

Initiator → Responder: DPP Status, SHA256(B_R), [SHA256(B_I),] { R-nonce }$_{k2}$

Otherwise the capabilities are compatible and the Initiator then uses the received public key(s) and the nonces to generate the remaining intermediate key(s) and the encryption key:

[$L = b_I * (B_R + P_R)$]

ke = HKDF(I-nonce | R-nonce, "DPP Key", M.x | N.x [| L.x])

As above, L is only computed and used in the computation of ke when doing mutual authentication.

k1 shall be irretrievably deleted upon processing of the DPP Authentication Response frame.

The Initiator unwraps the authenticating tag using the encryption key ke. If unwrapping returns FAIL, the Initiator aborts the exchange. Otherwise it generates a verifier:

R-auth' = H(I-nonce | R-nonce | P_I.x | P_R.x | [B_I.x |] B_R.x | 0)

Where 0 is a single octet with the value zero and where [B_I.x |] is only used in the computation of R-auth' when doing mutual authentication. If R-auth differs from R-auth' authentication fails and the Initiator shall respond with a DPP Authentication Confirm frame indicating failure by adding a hash of its bootstrapping key, a hash of the Initiator's bootstrapping key if mutual authentication was indicated in the DPP Authentication Response frame, the DPP Status field set to STATUS_AUTH_FAILURE, and wrapped data consisting of the Responder's nonce wrapped in k2:

Initiator → Responder: DPP Status, SHA256(B_R), [SHA256(B_I),] { R-nonce }$_{k2}$

The Initiator should alert the user on the authentication failure of the Responder. The Initiator aborts the exchange.

Otherwise, R-auth is identical to R-auth' and the Initiator authenticates the Responder. The Initiator then generates its own authenticating tag:

I-auth = H(R-nonce | I-nonce | P_R.x | P_I.x | B_R.x | [B_I.x |] 1)Where 1 is a single octet with a value one and where [B_I.x |] is only used in the computation of I-auth when doing mutual authentication. The I-auth is encoded in the Initiator Authenticating Tag attribute. The Initiator wraps the Initiator Authenticating Tag attribute with ke.

The Initiator then places a hash of Responder's public bootstrapping key, a hash of its public bootstrapping key if mutual authentication is being performed, DPP Status of STATUS_OK, and a Wrapped Data attribute containing the Initiator Authenticating Tag attribute into a DPP Authentication Confirm frame. The Initiator transmits this frame to the Responder.

Initiator → Responder: DPP Status, SHA256(B_R), [SHA256(B_I),] { I-auth }$_{ke}$

The Responder obtains the DPP Authentication Confirm frame and checks DPP Status. If DPP Status is STATUS_NOT_COMPATIBLE or STATUS_AUTH_FAILURE, the Responder unwraps the wrapped data portion of the frame using k2. If unwrapping fails, the Responder aborts the exchange. If unwrapping is successful, the Responder should alert the user on the reason for the protocol failure of the Initiator.

Otherwise, if DPP Status is STATUS_OK, the Responder unwraps the authenticating tag. If unwrapping returns FAIL, it aborts the exchange. Otherwise it generates a verifier, I-auth':

I-auth' = H(R-nonce | I-nonce | P_R.x | P_I.x | B_R.x | [B_I.x |] 1)

Where 1 is a single octet with a value one and where [B_I.x |] is only used in the computation of I-auth' when doing mutual authentication. If I-auth differs from I-auth' the Responder fails to authenticate the Initiator and aborts the exchange. Otherwise, it authenticates the Initiator. If authentication fails, this should result in an alert given to the user.

NOTE: unless mutual authentication is performed, the authentication of the Initiator to the Responder is weak and amounts to proof that it is an active participant in the exchange and knows the Responder's public key (in other words, it was acquired with some implied trust through a bootstrapping mechanism, for example one of those from section 5).

Upon completion of the DPP Authentication protocol, the secret elements M, N, and L if generated, and intermediate keys k1 and k2 shall be irretrievably deleted. The key, ke, can now be used as a pairwise symmetric key between the Initiator and Responder.

6.3 DPP Configuration protocol

6.3.1 Overview

Frames used in the DPP Configuration protocol use the Generic Advertisement Service (GAS) frame format (as defined in [2]) and include a series of fields, with a vendor specific Advertisement protocol ID. DPP specific attributes follow the GAS frame header. All DPP Configuration Protocol messages except for the Configuration Request with Fragments message are AES-SIV protected. AAD for use with AES-SIV for protected messages in the DPP Configuration protocol shall consist of all octets in the Query Request and Query Response fields up to the first octet of the Wrapped Data attribute, which is the last attribute in a DPP Configuration frame. When the number of octets of AAD would be zero, the number of components of AAD passed to AES-SIV is zero. When a DPP Configuration Response is fragmented, AES-SIV is applied to the full message prior to fragmentation. The fragmented message shall be reassembled before being decrypted.

The DPP Configuration protocol exchange shall immediately follow a successfully completed DPP Authentication protocol exchange with the Enrollee sending the DPP Configuration Request within one second of the completion of DPP Authentication. Both the Enrollee and the Configurator shall use the same MAC addresses and the same channel that was used during DPP Authentication protocol exchange.

6.3.2 DPP configuration request

Regardless of whether the Initiator or Responder took the role of Configurator, the DPP Configuration protocol is always initiated by the Enrollee. To start, the Enrollee generates a DPP Configuration Attributes object (see section 6.3.3) and generates a new nonce, E-nonce, whose length is determined according to Table 3. The E-nonce attribute and the DPP Configuration Attributesobject attribute are wrapped with ke. The wrapped attributes are then placed in a DPP Configuration Request frame, and sent to the Configurator.

Enrollee → Configurator: { E-nonce, configAttrib }$_{ke}$

6.3.3 DPP configuration response

The Configurator successfully receives the DPP Configuration Request frame and passes the ciphertext to AES-SIV with ke as the key. If AES-SIV returns FAIL, the Configurator aborts the exchange. Otherwise, it continues.

At this point, the Configurator has the Enrollee's attributes and the Enrollee's Network Access Key.

If the Configurator does not want to configure the Enrollee, for example if the Enrollee wishes to be enrolled as an AP and there are already enough APs in the network, the Configurator shall respond with a DPP Configuration Response indicating failure by adding the DPP Status field set to STATUS_CONFIGURE_FAILURE, and wrapped data consisting of the Enrollee's nonce wrapped in ke:

Configurator → Enrollee: DPP Status, { E-nonce }$_{ke}$

Otherwise, the Configurator uses the attributes supplied by the Enrollee to construct a confirmation message consisting of a DPP Configuration object (see section 6.3.6) and the received E-nonce. This message is wrapped with ke. The ciphertext output by AES-SIV is then, together with a DPP Status field set to STATUS_OK, copied into a DPP Configuration Response frame and sent to the Enrollee.

Configurator → Enrollee: DPP Status, { E-nonce, configurationObject }$_{ke}$

Upon successful receipt of the DPP Configuration Response frame, the Enrollee unwraps the received ciphertext with ke. If AES-SIV returns FAIL or the configuration cannot be validated, the Enrollee may resend a DPP Configuration Request or it may unsuccessfully terminate configuration. Otherwise, if the received DPP Status is not STATUS_OK or the received E-nonce differs from the one sent to the Configurator, the Enrollee may resend a DPP Configuraiton Request or it may unsuccessfully terminate configuration. If the received configurationObject has "akm=dpp", the Enrollee verifies that the networkAccessKey in the Connector is the same as the protocol key it used in the preceding DPP Authentication exchange. If not, the Enrollee unsuccessfully terminates configuration. Otherwise, it provisions the network with the received DPP Configuration object, stores the Configurator identity and signature key, populates its cache of Connectors or other network credentials, and, optionally, appends the indicated configurators onto the appropriate linked lists. The received configuration applies to all radios of the device (for example., a dual-band dual-concurrent AP as the Enrollee uses the received Connector on all radios operating the provisioned network).

6.3.4 DPP Configuration Attributes object

6.3.4.1 Overview

The DPP Configuration Attributes object is a JSON (JavaScript Object Notation) encoded data structure, see RFC 7159 [13], that is transmitted as an attribute in the DPP Configuration Request frame.

JSON encodes data as a series of data types (strings, numbers, Booleans, and null) and structure types (objects and arrays), formatted as name/value pairs.

6.3.4.2 Encoding

The JSON encoding for the DPP Configuration Attributes object is defined in Table 12.

Table 12. DPP Configuration Attributes object

Parameter	Name	Type	Value	Description
DPP Configuration Attributes object	configAttrib	OBJECT		
Device Name	name	STRING		
Wi-Fi Technology	wi-fi_tech	STRING	infra	The type of network that the Enrollee wishes to be enrolled in.
Network Role	netRole	STRING	sta, ap	The role that the Enrollee wishes to obtain.

An example of a JSON encoded DPP Configuration Attributes is given in Figure 10.

```
{
 "name":"My Device",
 "wi-fi_tech":"infra",
 "netRole":"sta"
}
```

Figure 10. Example DPP Configuration Attributes object

6.3.5 Connector

6.3.5.1 Overview

The Configurator possesses a signing key pair (c-sign-key, C-sign-key). The c-sign-key is used by the Configurator to sign Connectors, whereas the C-sign-key is used by provisioned devices to verify Connectors of other devices are signed by the same Configurator. The Configurator signs the Connector according to the procedures described in section 4.2. The procedures to compute the digital signature of a Connector and the procedure to verify such signature are described in FIPS-186-4 [20] and are specified in section 4.2. Connectors signed with the same c-sign-key manage connections in the same network.

The Configurator sets the public analog of the enrollee[3] protocol key as the netAccessKey in the Connector data structure, and assigns DPP Connector attribute depending on the peer devices that the enrollee will be provisioned to connect.

6.3.5.2 Encoding

The JSON encoding for the DPP Connector attribute body object is given in Table 13.

Table 13. DPP Connector attribute body object format

Parameter	Name	Type	Value	Description
DPP Connector Body object	dppCon	OBJECT		
	groups[1]	ARRAY		The groups array comprises an array of group objects
	groupId	STRING	Freeform string or wildcard set to "*"	A string identifying the identity of the group in the group object. The owner of this Connector is allowed by the Configurator to connect to devices in this group.
	netRole	STRING	sta, ap	The role assigned to the owner of this Connector in this group.
	netAccessKey	JWK		JSON web key with encoding defined in RFC 7517 [16]; "key_ops" and "use" objects in a "netAccessKey" object shall not be present. Note that JSON Web Keys use an uncompressed format.
	expiry	STRING		Timestamp for net access key expiry, see section 4.3.3.4. Optionally present.
Notes: 1. At least one group object shall be present in the groups object.				

An example of a JSON encoded DPP Connector Body Object is given in Figure 11.

```
{
 "groups":
 [
   {"groupId":"home","netRole":"sta"},
```

[3] Note that the enrollee protocol key will either be the initiator Protocol key, PI, or the Responder Protocol key, PR depending on the device that initiates the DPP Authentication sequence being the Enrollee or Configurator.

```
      {"groupId":"cottage","netRole":"sta"}
    ],
    "netAccessKey":
    {
      "kty":"EC",
      "crv":"P-256",
      "x":"Xj-zV2iEiH8XwyA9ijpsL6xyLvDiIBthrHO8ZVxwmpA",
      "y":"LUsDBmn7nv-LCnn6fBoXKsKpLGJiVpY_knTckGgsgeU"
    },
    "expiry":"2019-01-31T22:00:00+02:00"
  }
```

Figure 11. Example DPP Connector Body object

The object is signed and encoded as a JWS compact serialization as described in RFC 7515 [15]. The JWS Protected Header for the DPP Connector Body object shall contain the "typ", "kid" and "alg" objects. The "typ" value shall be “dppCon”, denoting the DPP Connector Body object, see section 6.3.5 and Table 13. The value of "kid" shall be the base64url encoded SHA256 hash of the uncompressed form, see [6] ANSI X9.63, of the public signing key of the configurator. An example of a JWS Protected Header using the public signing key used in the example DPP Configuration object in Figure 14 is shown in Figure 12.

```
{
  "typ":"dppCon",
  "kid":"kMcegDBPmNZVakAsBZOzOoCsvQjkr_nEAp9uF-EDmVE",
  "alg":"ES256"
}
```

Figure 12. Example JWS protected header

An example DPP Connector, the JWS compact serialization of the JWS of the DPP Connector Body object "dppCon" of Figure 11 with all whitespace removed, using the JWS Protected Header of Figure 12 with all white space removed, is shown in Figure 13.

```
eyJ0eXAiOiJkcHBDb24iLCJraWQiOiJrTWNlZ0RCUG1OWlZha0FzQlpPek9vQ3N2UWprcl9uRUFwOXVGLUVEbVZFIiwiYWxnIjo
iRVMyNTYifQ
.
eyJncm91cHMiOlt7Imdyb3VwSWQiOiJob21lIiwibmV0Um9sZSI6InN0YSJ9LHsiZ3JvdXBJZCI6ImNvdHRhZ2UiLCJuZXRSb2x
lIjoic3RhIn1dLCJuZXRBY2Nlc3NLZXkiOnsia3R5IjoiRUMiLCJjcnYiOiJQLTI1NiIsIngiOiJYai16VjJpRWlIOFh3eUE5aW
pwc0w2eHlMdkRpSUJ0aHJITzhaVnh3bXBBIiwieSI6IkxVc0RCbW43bnYtTENubjZmQm9YS3NLcExHSmlWcFlfa25UY2tHZ3NnZ
VUifSwiZXhwaXJ5IjoiMjAxOS0wMS0zMVQyMjowMDowMCswMjowMCJ9
.
8fJSNCpDjv5BEFfmlqEbBNTaHx2L6c_22Uvr9KYjtAw88VfvEUWiruECUSJCUVFqv1yDEE4RJVdTIw3aUDhlMw
```

Figure 13. Example DPP Connector (with line breaks for display purposes only)

6.3.6 DPP Configuration object

6.3.6.1 Overview

The DPP Configuration object is a JSON (JavaScript Object Notation) encoded data structure, see RFC 7159 [13], that is transmitted as an attribute in the DPP Configuration Response frame.

The details of the DPP Configuration object data structure are found in section 4.3.

JSON encodes data as a series of data types (strings, numbers, Booleans, and null) and structure types (objects and arrays), formatted as name/value pairs.

Binary data used in the DPP Configuration object such as keys and digital signatures, is encoded using base64url (see RFC 4648 [14]). The JWT (JSON Web Token) procedures described in RFC 7517 [16] are used to encode keying material.

The DPP Configuration object parameters are given in Table 14.

Table 14. DPP Configuration object parameters

Parameter	Name	Type	Value	Description
DPP Configuration object	configurationObject	OBJECT		
Wi-Fi Technology object:	wi-fi_tech	STRING	infra	Future revisions may include the values dpp_config, nan, p2p, asp2.
Service	svc	STRING		Optional parameter depending on the value of wi-fi_tech
Discovery object:	discovery	OBJECT		
SSID	ssid	STRING	alpha numeric	The name of the network to connect to
Credential object	cred	OBJECT		
Authentication and key management type	akm	STRING	psk, dpp, sae, psk+sae	The authentication type for the network
Pre-shared key	psk_hex	STRING		Pre-shared key encoded in hex. Conditionally present for akm=psk.
WPA2 Passphrase and/or SAE password	pass	STRING		PSK or SAE Passphrase/password. Conditionally present for akm=psk, sae, psk+sae.
DPP Connector	signedConnector	STRING		DPP Connector as a JWS, see section 4.2. Present for akm=dpp.
C-sign-key	csign	JWK		Configurator public key (configurator); "key_ops" and "use" objects in a "csign" object shall not be present. Present for akm=dpp. Note that JSON Web Keys use an uncompressed format.

6.3.6.2 Encoding

The JSON encoding of the DPP Configuration object described in section 4.3 is given in Table 14.

An example of a JSON encoded DPP Configuration object is given in Figure 14.

```
{
 "wi-fi_tech":"infra",
 "discovery":
 {
    "ssid":"mywifi"
 },
 "cred":
 {
    "akm":"dpp",
    "signedConnector":
"eyJ0eXAiOiJkcHBDb24iLCJraWQiOiJrTWNlZ0RCUG1OWlZha0FzQlpPek9vQ3N2UWprcl9uRUFwOXVGLUVEbVZFIiwi
YWxnIjoiRVMyNTYifQ.eyJncm91cHMiOlt7Imdyb3VwSWQiOiJob21lIiwibmV0Um9sZSI6InN0YSJ9LHsiZ3JvdXBJZC
I6ImNvdHRhZ2UiLCJuZXRSb2xlIjoic3RhIn1dLCJuZXRBY2Nlc3NLZXkiOnsia3R5IjoiRUMiLCJjcnYiOiJQLTI1NiI
sIngiOiJYai16VjJpRWlIOFh3eUE5aWpwc0w2eHlMdkRpSUJ0aHJITzhaVnh3bXBBIiwieSI6IkxVc0RCbW43bnYtTENu
bjZmQm9YS3NLcExHSmlWcFlfa25UY2tHZ3NnZVUifSwiZXhwaXJ5IjoiMjAxOS0wMS0zMVQyMjowMDowMCswMjowMCJ9.
8fJSNCpDjv5BEFfmlqEbBNTaHx2L6c_22Uvr9KYjtAw88VfvEUWiruECUSJCUVFqv1yDEE4RJVdTIw3aUDhlMw",
    "csign":
    {
       "kty":"EC",
       "crv":"P-256",
       "x":"MKBCTNIcKUSDii11ySs3526iDZ8AiTo7Tu6KPAqv7D4",
       "y":"4Etl6SRW2YiLUrN5vfvVHuhp7x8PxltmWWlbbM4IFyM",
       "kid":"kMcegDBPmNZVakAsBZOzOoCsvQjkr_nEAp9uF-EDmVE"
    }
```

```
  }
}
```

Figure 14. Example DPP Configuration object

6.4 Network introduction protocol

6.4.1 Introduction

A device that has finished the enrollment portion of DPP is able to communicate to select peers on the network controlled by the Configurator. In infrastructure networks, only the STA shall initiate the DPP Network Introduction protocol. When establishing a connection to a specific known device, for example, an AP in an infrastructure BSS advertising the DPP AKM suite selector in Beacon and Probe Response frames, the DPP Peer Discovery Request frame shall be sent as a unicast frame on the operating channel of the recipient. Otherwise, the device discovers peers on the network by broadcasting DPP Peer Discovery Request frames containing a single octet transaction identifier and its $Connector_B$. The transaction identifier is a one octet value uniquely identifying a pending transaction.

device B → device A: Transaction ID, $Connector_B$

A device A that receives a DPP Peer Discovery Request frame from a device B shall decode and check the Connector.

If device A finds any problem in the received Connector, for example

- a syntax error in the received Connector such as a misspelled field name,
- a wrong or missing value in the received Connector such as a missing kid field, erroneous NAK expiry value
- an expired NAK in the Connector,
- a failing signature verification if device A possesses the matching C-sign-key,

it sends a DPP Peer Discovery Response framewith STATUS_INVALID_CONNECTOR and the received Transaction ID:

device A → device B: Transaction ID, DPP Status

If device A did not find any problem in the received Connector, and does not possess the matching C-sign-key for checking its signature, it sends a DPP Peer Discovery Response frame with STATUS_NO_MATCH and the received Transaction ID:

device A → device B: Transaction ID, DPP Status

If device A finds the incoming Connector fully correct and has verified that its signature is correct, meaning device A does possess the matching C-sign key, device A tries to find a matching Connector by extracting device B's netAccessKey, PK, and checks whether it is permitted by a Configurator to establish a link to device B, with a Connector containing a matched groupId, see section 6.4.2.

If it has not been permitted to establish a link to this device, or if device B's netAccessKey, PK, in all matching Connectors has expired, it sends a DPP Peer Discovery Response frame with STATUS_NO_MATCH and the received Transaction ID:

device A → device B: Transaction ID, DPP Status

If the recipient is permitted to speak to the peer, it derives a shared secret N, using the private analog to its netAccesskey, nk, and the peer's public network access provisioning key PK derived from the $Connector_B$, a PMK, and a PMKID using both public keys, NK and PK:

N = nk * PK

PMK = HKDF(<>, "DPP PMK", N.x)

PMKID = Truncate-128(SHA256(min(NK.x, PK.x) | max(NK.x, PK.x)))

Upon generation of the PMK and PMKID, N shall be irretrievably deleted. Device A then shall respond with a DPP Peer Discovery Response frame consisting of DPP_STATUS_OK, the transaction identifier copied from the DPP Peer Discovery Request frame and a matching $Connector_A$.

device A → device B: Transaction ID, DPP Status, $Connector_A$

Device B shall drop a received DPP Peer Discovery Response frame with an unknown Transaction ID. Transaction ID is used by device B to match its pending request and netAccessKey.

Device B shall check the $Connector_A$ from device A in the same way as described above for device A. If device B has not been permitted to establish a link to device A, it shall drop the DPP Peer Discovery Response frame.

If device B verifies that it is permitted to establish a link to device A, device B(and new Peer) derives a shared secret N using the private analog to its network access provisioning key, pk, and device A's network access provisioning key NK, a PMK, and a PMKID:

N = pk * NK

PMK = HKDF(<>, "DPP PMK", N.x)

PMKID = Truncate-128(SHA256(min(PK.x, NK.x) | max(PK.x, NK.x)))

Upon generation of the PMK and PMKID, N shall be irretrievably deleted.

NOTE: sending a DPP Peer Discovery Response frame with STATUS_INVALID_CONNECTOR means that there is a configuration problem in device A and that sending a DPP Peer Discovery Response frame with STATUS_NO_MATCH means that there is a configuration problem in device B.

6.4.2 Connector group comparison

A Connector received by a peer device may contain multiple group attributes. The peer device evaluates these attributes to determine whether it shall initiate a connection to the device that owns the received Connector.

When group attributes are contained in the Connector, the peer device compares the received groupId(s) to a list of its Connector groupId(s). If a groupId matches, the device evaluates the netRole in the same group attribute to confirm that the netRole is compatible with its netRole for the matching group attribute as specified in Table 15. If there is no groupId match or the netRole is not compatible, the device does not have a group attribute match.

If no group attribute match has been found, the device shall not initiate a connection to the device that owns the received Connector. However, a Peer Discovery Response shall be sent with the error code STATUS_NO_MATCH for DPP_STATUS, in case the Connector is received in a Peer Discovery Request.

Table 15. netRole Compatibility

netRole of matching groupId in received Connector	netRole of device's matching Connector groupId	Compatibility
ap	ap	Not compatible
ap	sta	Compatible
sta	ap	Compatible
sta	sta	Not compatible

6.5 Network access protocols

The two peers shall establish an association using procedures defined in [2]. If this DPP Network Introduction protocol is used to derive the PMK, the peers shall use the AKM defined in section 8.4.2. Alternatively, if the Network Introduction protocol is not used, the peers may establish a Security Association through any similar Wi-Fi protocol which uses a shared key as a base.

7 State machines

7.1 Initiator state machine

The Initiator maintains a parent process that creates instances of the state machine, passes state machine events, and transmits frames that are output by the state machine.

7.1.1 States

The states of each Initiator instance are:

- Nothing—the initial state of an Initiator protocol instance.
- Bootstrapped—a peer's public key has been obtained and trusted.
- Authenticating—the Initiator is trying to authenticate the peer.
- Authenticated—the Initiator has authenticated the peer.

7.1.2 Events and output

State machine instances receive events. Received events advance the state machine and optionally produce output.

The events received by an Initiator instance are:

- Bootstrapping—one of the bootstrapping techniques has been accomplished and trust in a peer's public key has been obtained.
- Trigger—start the authentication and provisioning process.

Events generated by an Initiator instance are:

- NoPeer—discovery of the peer has been abandoned.
- BadAuth—the peer could not be authenticated or provisioned.

The outputs an Initiator state machine produces are:

- "-"—the NULL response, no response at all.
- AuthReq—a DPP Authentication Request frame.
- AuthConf—a DPP Authentication Confirm frame.

7.1.3 Variables

Each instance of the Initiator uses variables to facilitate running the state machine. These are:

- t—a timer that counts down a certain number of seconds and whose expiry is be detected by the state machine.
- Sent—a counter to indicate how many times a frame has been sent in one state.

7.1.4 Parent process behavior

The parent process creates Initiator instances to manage the authentication and provisioning process for an individual peer. It passes frames received by the peer or external inputs such as a button press from a UI indicating a desired action.

7.1.5 State machine behavior

The parent process creates an Initiator instance with the state set to Nothing. The variables Sent is set to zero, and t is created.

The Initiator state machine is graphically illustrated in Figure 15.

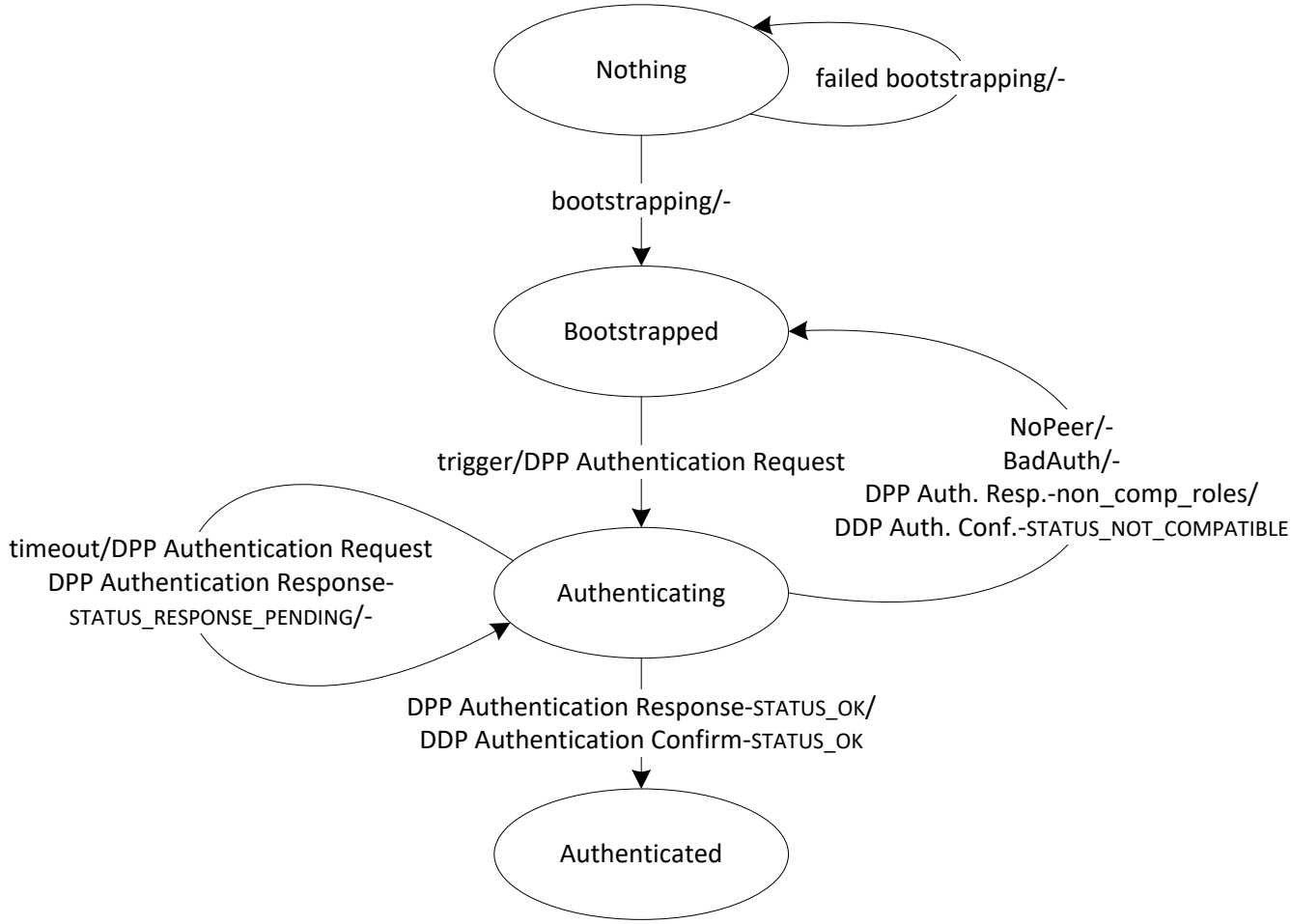

Figure 15. Initiator State Machine

Nothing state

All events in the Nothing state are ignored by the Initiator except bootstrapping. The parent process sends a bootstrapping event to a state machine instance upon completion of one of the bootstrapping techniques. Upon receiving a bootstrapping event the state machine advances to the Bootstrapped state; it produces no output. A byproduct of bootstrapping is a peer's trusted public key.

Bootstrapped state

In the Bootstrapped state, the Initiator ignores all events except trigger. The parent process sends a trigger event when it wants the Initiator's state machine to proceed with authentication of a bootstrapped peer (for example, if a button on a UI is pressed). Upon receipt of a trigger event, t is set to 10 seconds, Sent is set to one, a DPP Authentication Request frame is sent to the peer identified by the bootstrapped public key, and the state machine transitions to the Authenticating state. A failed bootstrapping attempt will bring the state machine back to the Nothing state.

Authenticating state

In the Authenticating state, the Initiator receives DPP Authentication Response frames and timer events. All other events are ignored.

If a DPP Authentication Response frame is received and is well-formed, has STATUS_OK and if the Responder's Capability pairs correctly to that of the Initiator, then t is cancelled and reset to 10 seconds, Sent is set to one, a DPP Authentication Confirm frame with STATUS_OK is sent to the peer, and the state machine transitions to the Authenticated state.

If a DPP Authentication Response frame is received and is well-formed, has STATUS_OK and if the Responder's Capability is the same as that of the Initiator, then t is cancelled, Sent is set to zero, a DPP Authentication Confirm frame with STATUS_NOT_COMPATIBLE is sent to the Responder, and the state machine transitions to the Bootstrapped state.

If a DPP Authentication Response frame is received and is well-formed and has STATUS_RESPONSE_PENDING, t is cancelled and reset to 120 seconds and Sent is set to six.

If a DPP Authentication Response frame is received that is malformed, t is cancelled, Sent is set to 0, a BadAuth event is generated, and the state machine transitions to the Bootstrapped state.

If t expires and an ACK frame was received for the previously transmitted DPP Authentication Request frame, a NoPeer event is generated and the state machine transitions to the Bootstrapped state. Otherwise, if t expires, the state machine checks the value of Sent. If Sent is five or less Sent is incremented, then t is set to 10 seconds, the DPP Authentication Request frame is resent, and the state machine remains in the Authenticating state. If Sent is greater than five, Sent is set to zero, a NoPeer event is generated, and the state machine transitions to the Bootstrapped state.

Authenticated

The Authenticated state is the terminal state for an Initiator instance. If the Initiator is a Configurator, the state machine sets timer t to 10 seconds and awaits the start of provisioning (see section 7.3); otherwise it remains in the Authenticated state awaiting a trigger event to begin the provisioning process (see section 7.4).

7.2 Responder state machine

DPP capable devices maintain a parent process that is capable of creating instances of the DPP Responder state machine.

The parent process delivers DPP frames to the Responder state machine, sends events, and processes indications to send DPP frames to peers engaged in the DPP protocol.

7.2.1 States

The states of each Responder instance are:

- Bootstrap Key Acquiring—the state machine is in the process of acquiring the bootstrapping key from the Initiator through user interaction.
- Awaiting—a quiescent state.
- Authenticating—the state machine is in the process of authenticating an Initiator.
- Authenticated— the state machine has finished normally.

7.2.2 Events and output

The Responder state machine receives authentication requests, authentication confirmations and timer events.

Events generated by a Responder protocol instance are:

- BadAuth—the Initiator could not be authenticated or provisioning failed.

The outputs a Responder state machine produces are:

- “-“—the NULL response, no response at all.
- AuthResp—a DPP Authentication Response frame.

7.2.3 Variables

Each protocol instance of the Responder state machine uses variables to facilitate its operation. These are:

- t—a timer that counts down a certain number of seconds and whose expiry is detected by the state machine.
- Sent—a counter to indicate how many times a frame has been sent in one state.

7.2.4 State machine behavior

A Responder instance is created in the Awaiting state. Sent is set to zero, and t is created.

The Responder state machine is graphically illustrated in Figure 16.

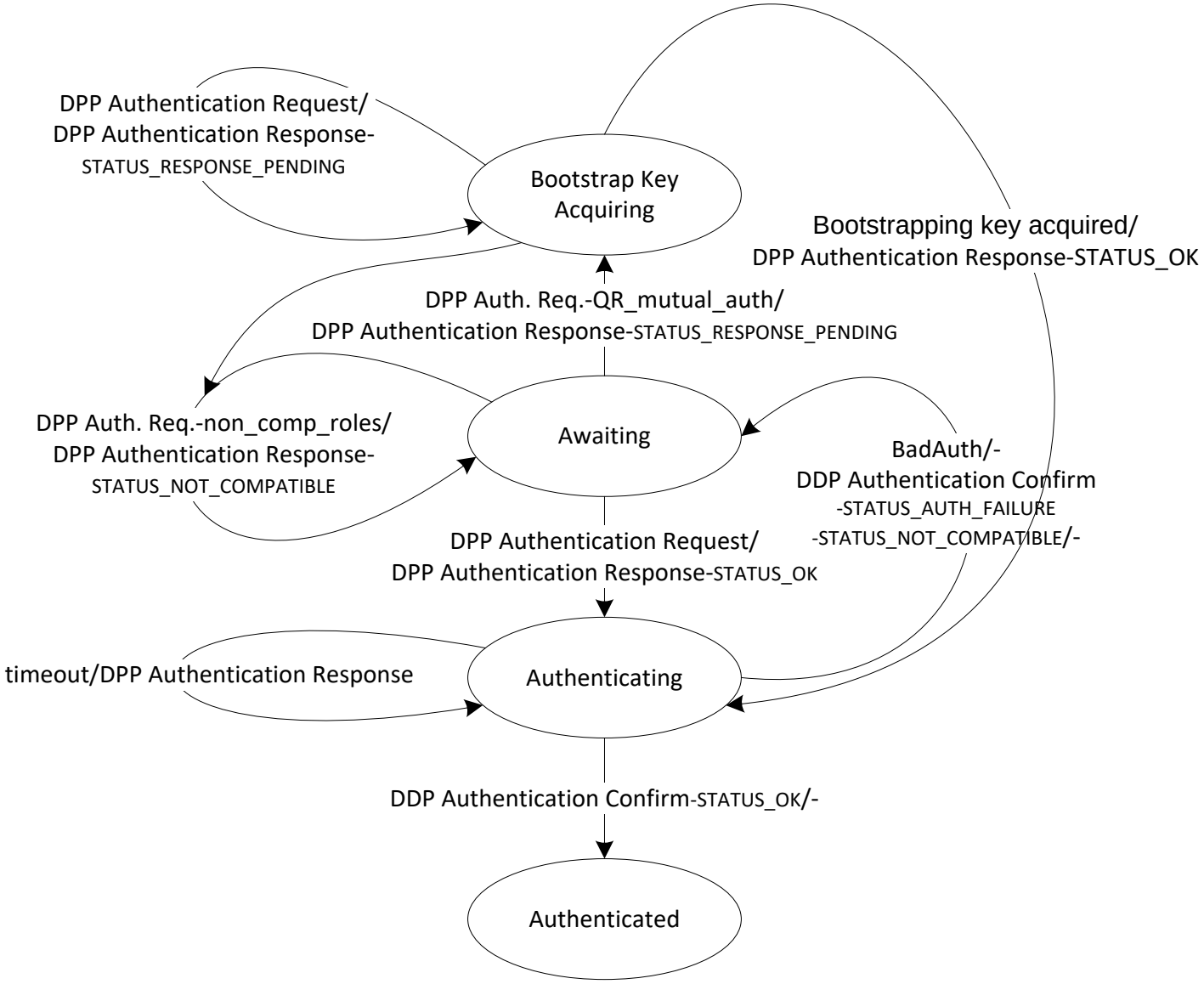

Figure 16. Responder state machine

Awaiting state

This is a quiescent state in which the Responder awaits contact by an Initiator. DPP Authentication Request frames are received in the Awaiting state. All other events are ignored.

Malformed DPP Authentication Request frames are ignored. Well-formed DPP Authentication Request frames result in the generation of data necessary to engage in the DPP Authentication protocol.

If a DPP Authentication Request frame is received that is well-formed and if the Initiator's Capability is the same as that of the Responder, a DPP Authentication Response frame with STATUS_NOT_COMPATIBLE is sent to the peer that sent the Request and the state machine remains in the Awaiting state.

The following holds for a Responder that does not have to do mutual authentication or that has to do mutual Authentication and has already obtained the Initiator's Bootstrapping key. If a DPP Authentication Request frame is received that is well-formed and if the Initiator's Capability is complementary to that of the Responder, then a DPP Authentication Response frame with STATUS_OK is sent to the peer that sent the Request, Sent is set to one, t is set to 10 seconds, and the state machine transitions to the Authenticating state.

The following holds for a Responder that has to do mutual authentication and that has not yet obtained the Initiator's Bootstrapping key. If a DPP Authentication Request frame is received that is well-formed and if the Initiator's Capability is complementary to that of the Responder, then a DPP Authentication Response frame with STATUS_RESPONSE_PENDING is sent to the peer that sent the Request, and the state machine transitions to the Bootstrap Key Acquiring state.

Bootstrap Key Acquiring state

In the Bootstrap Key Acquiring state, the state machine receives DPP Authentication Request frames and bootstrap key acquired events.

The user is instructed to acquire the bootstrapping key of the Initiator (for example, scan its QR-code).

If the bootstrapping key of the Initiator is acquired successfully and if it is well-formed, then a DPP Authentication Response frame with STATUS_OK is sent to the peer that sent the Request, Sent is set to one, t is set to 10 seconds, and the state machine transitions to the Authenticating state.

If a DPP Authentication Request frame is received from the same Initiator from which the Responder is acquiring the bootstrapping key and if furthermore that message is well-formed and if the Initiator's Capability is complementary to that of the Responder, then a DPP Authentication Response frame with STATUS_RESPONSE_PENDING is sent to the peer that sent the Request, and the state machine remains in the Bootstrap Key Acquiring state.

If a DPP Authentication Request frame is received from the same Initiator from which the Responder is acquiring the bootstrapping key and if furthermore that message is well-formed and if the Initiator's Capability is the same as that of the Responder, then a DPP Authentication Response frame with STATUS_NOT_COMPATIBLE is sent to the peer that sent the Request and the state machine transitions to the Awaiting state.

DPP Authentication Request frames from other Initiators are ignored.

Authenticating state

In the Authenticating state, the state machine receives DPP Authentication Confirm frames and timer events.

If a DPP Authentication Confirm frame is received and it is well-formed and has STATUS_OK, then t is cancelled and the state machine transitions to the Authenticated state.

If a DPP Authentication Confirm frame is received and it is well-formed and has STATUS_AUTH_FAILURE or STATUS_NOT_COMPATIBLE, then t is cancelled and the state machine transitions to the Awaiting state.

If a DPP Authentication Confirm frame is received and it is malformed, then t is cancelled, Sent is set to zero, a BadAuth event is generated and the Responder transitions to the Awaiting state.

If t expires and an ACK frame was received for the previously transmitted DPP Authentication Response frame, a BadAuth event is generated and the state machine transitions to the Awaiting state. Otherwise, if t expires, the value of Sent is checked. If Sent is five or less, then Sent is incremented, t is set to 10 seconds, the DPP Authentication Response frame is sent, and the Responder remains in the Authenticating state. If Sent is greater than five, a BadAuth event is generated and the state machine transitions to the Awaiting state.

If any other frame other than a DPP Authentication Confirm frame is received, the frame shall be dropped and the Responder remains in the Authenticating state.

Authenticated state

If the Responder is a Configurator, the state machine sets timer t to 10 seconds and awaits the start of provisioning (see section 7.3), otherwise it remains in the Authenticated state awaiting a trigger event to begin the provisioning process (see section 7.4).

7.3 Configurator state machine

When a device is acting in the role of Configurator, the parent process creates a Configurator State machine following the successful completion of an Initiator or Responder state machine. The former state machines successfully end in the Authenticated state and it is from this state that the Configurator State machine begins.

The parent process delivers DPP frames to the Configurator state machine, sends it events, and processes configuration responses to send DPP frames to peers engaged in the DPP protocol.

7.3.1 States

The states of each Configurator instance are:

- Authenticated—the initial state of a Configurator protocol instance.
- Finished—the state machine successfully terminated.
- Terminated—a failure state indicating unsuccessful provisioning of an authenticated peer.

7.3.2 Events and output

State machine instances receive events. Received events advance the state machine and optionally produce output.

The Configurator receives configuration requests.

Events generated by a Configurator are:

- NoPeer—provisioning has been abandoned because the Enrollee has gone away.

The outputs a Configurator state machine produces are:

- “-“—the NULL response, no response at all.
- ConfResp—a DPP Configuration Response.

7.3.3 Variables

Each Configurator state machine uses the following variables:

- t—a timer that counts down a certain number of seconds and whose expiry is detected by the state machine.

7.3.4 Parent process behavior

The parent process creates Configurator instances to manage provisioning a peer after successful authentication. It passes frames received by the peer to the Configurator state machine.

7.3.5 State machine behavior

The parent process creates a Configurator with its initial state set to Authenticated. Timer t is created and set to 10 seconds.

The Configurator state machine is graphically illustrated in Figure 17.

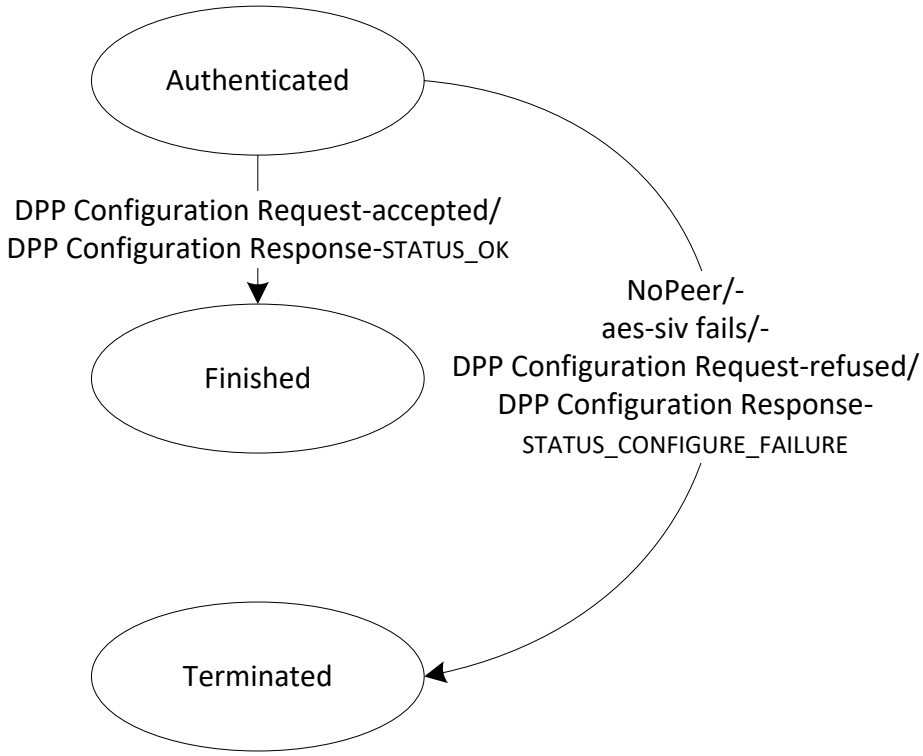

Figure 17. Configurator state machine

Authenticated state

If t expires, then a NoPeer event is generated and the Configurator transitions to the Terminated state.

If a DPP Configuration Request is received for which AES-SIV decryption fails, or which contains the wrong value of E-nonce, then the state machine transitions to the Terminated state.

If a correct DPP Configuration Request message is received and the Configurator wants to grant the request, then a DPP Configuration Response frame with STATUS_OK is generated and the state machine transitions to the Finished state.

If a correct DPP Configuration Request message is received and the Configurator does not want to grant the request, then a DPP Configuration Response with STATUS_CONFIGURE_FAILURE is generated and the state machine transitions to the Terminated state.

If any other frame other than a DPP Configuration Request frame is received or the DPP Configuration Request frame is malformed, the frame shall be dropped and the state machine remains in the Authenticated state.

Finished state

This represents a successful terminus for the state machine. An indication of success may be generated. The parent process may perform any housekeeping at this point.

Terminated state

This represents an unsuccessful terminus for the state machine. The parent process may perform any housekeeping at this point.

7.4 Enrollee state machine

When a device is acting as an Enrollee, the parent process creates an Enrollee State machine following the successful completion of an Initiator or Responder state machine. The former state machines successfully end in the Authenticated state and it is from this state that the Enrollee State machine begins.

The parent process delivers DPP frames to the Enrollee, sends it events, and sends DPP frames from the Enrollee to its peers engaged in the DPP protocol.

7.4.1 States

The states of each Enrollee instance are:

- Authenticated—the state machine is starting the process of provisioning.
- Provisioning—the state machine is in the process of being provisioned by a Configurator.
- Provisioned—the state machine has finished normally.
- Terminated—the state machine has finished abnormally or has been terminated.

7.4.2 Events and output

The events received by an Enrollee are:

- Trigger—start the provisioning process.

Events generated by the Enrollee are:

- NoPeer—the authenticated Configurator failed to provision the Enrollee.

The outputs an Enrollee produces are:

- "-"—the NULL response, no response at all.
- ConfReq—a DPP Authentication Configuration request.

7.4.3 Variables

Each protocol instance of the Enrollee state machine uses the following variables:

- t—a timer that counts down a certain number of seconds and whose expiry is detected by the state machine.
- Sent—a counter to indicate how many times a frame has been sent in one state.

7.4.4 State machine behavior

An Enrollee state machine is created in the Authenticated state. Sent is set to zero, and t is created. The Enrollee state machine is graphically illustrated in Figure 18.

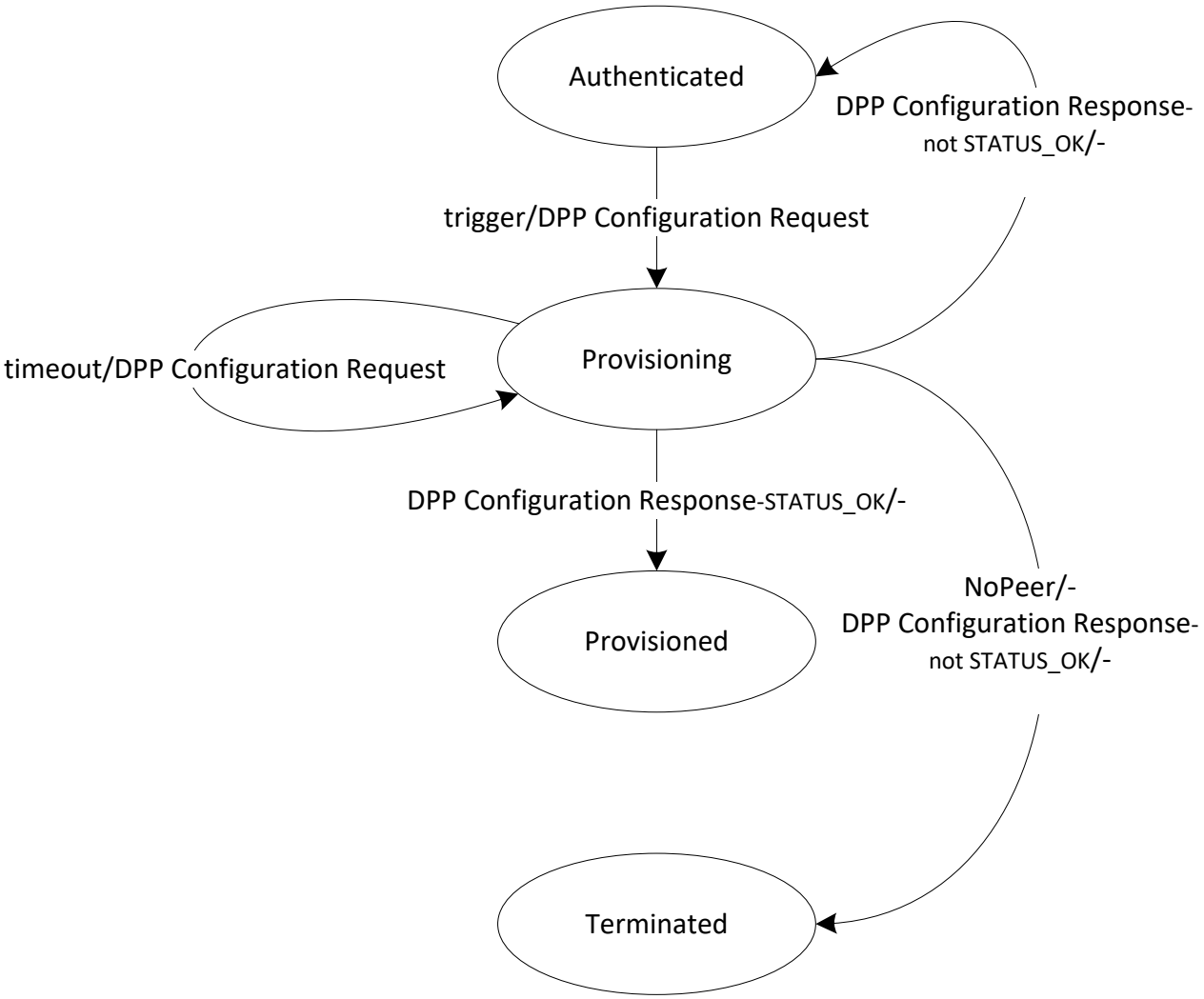

Figure 18. Enrollee state machine

Authenticated state

In the Authenticated state, the Enrollee ignores all events except a trigger. The parent process sends a trigger event when it wants the state machine to proceed with provisioning. Upon receipt of a trigger, t is set to 10 seconds, Sent is set to one, a DPP Configuration Request is sent to the peer and the state machine transitions to the Provisioning state.

Provisioning state

In the Provisioning state, the Enrollee receives DPP Configuration Response frames and timer events. All other events are ignored.

If a DPP Configuration Response frame is received and is well-formed, has STATUS_OK, has the correct E-nonce and the signature verification was successful, then t is cancelled and the state machine transitions to Provisioned.

If any other DPP Configuration Response frame than the one described in the line above is received, then t is cancelled and the state machine transitions to Authenticated or to the Terminated state (implementers option).

If t expires and an ACK frame was received for the previously transmitted DPP Configuration Request frame, the Enrollee generates a NoPeer event and transitions to the Terminated state. Otherwise, if t expires, the state machine checks the value of Sent. If Sent is five or less, then Sent is incremented, t is set to 10 seconds, the DPP Configuration Request frame is resent, and the state machine remains in the Provisioning state. If Sent is greater than five, the Enrollee generates a NoPeer event and transitions to the Terminated state.

If any other frame other than a DPP Configuration Response is received, the frame shall be dropped and the Enrollee remains in the Provisioning state.

Provisioned state

This state indicates a successful terminus for the state machine. An indication of success may be generated. The parent process may perform any housekeeping at this point.

Terminated state

This state indicates an unsuccessful terminus for the state machine. An indication of failure may be generated. The parent process may perform any housekeeping at this point.

7.5 Detailed protocol description

When a device is setup as a Configurator, it generates the key pair (c-sign-key, C-sign-key), to sign and verify Connectors, respectively.

An Initiator starts in the Nothing state. When the Responder begins, it starts in the Awaiting state.

Either a Configurator or an Enrollee can initiate DPP bootstrapping or authentication exchange. The other device will respond. In other words, it becomes the Responder.

7.5.1 DPP bootstrapping

The Initiator bootstraps the Responder to obtain the Responder's bootstrap key (Br). Optionally the Responder bootstraps the Initiator. The bootstrapping methods are described in section 5.

The Initiator transits from the state the Nothing state to the Bootstrapped state. The Responder state machine remains in the Awaiting state.

7.5.2 DPP authentication exchange

A DPP Authentication exchange takes place between an Authenticating Initiator and an Awaiting Responder. Note that only one main flow is described here and that section 6.2.2 and section 6.2.3 specify several alternative flows that depend on the capabilities of Initiator and Responder being non-complementary, or the Responder still having to acquire the bootstrapping key of the Initiator.

The Initiator first prepares the following elements (see section 6.2.2):

1. Sets I-capabilities to indicate whether it is a Configurator or Enrollee.
2. Generates an Initiator protocol key pair - (P_I/p_I)), which is either the network access key when the Initiator is an Enrollee or an ephemeral key when it is a Configurator.
3. Computes k1 using the Responder's bootstrapping key (B_R) and the Initiator's private protocol key (p_I).
4. Generates I-nonce.

The Initiator then transmits an Authentication Request frame to the Responder. The message includes (see section 6.2.2):

1. A hash of the Responder's bootstrapping key (B_R).
2. A hash of the Initiator's bootstrapping key (B_I).
3. The Initiator's public protocol key (P_I).
4. I-nonce concatenated with I-capabilities and wrapped with AES-SIV, using k1 and the AAD.

Upon receipt of an Authentication Request frame, the Responder (see section 6.2.3):

1. Transits from the Awaiting state to the Authenticating state.
2. Computes k1 using the Initiator's public key and the Responder's private bootstrapping key.

3. Unwraps the I-nonce and the I-capabilities with AES-SIV, using k1 and the AAD.
    a. If AES-SIV fails, the Responder aborts the exchange.
    b. If the Responder cannot negotiate its capabilities, given I-capabilities, it also aborts the exchange.
    c. In either case, the Responder transits from the Authenticating state back to the Awaiting state.

The Responder then prepares the following data (see section 6.2.3):

1. Sets R-capabilities to indicate that it is either the Enrollee or the Configurator, whichever is opposite of the role the Initiator declared.
2. Generates the Responder's protocol key pair (P_R/p_R), which is either the network access key when the Responder is an Enrollee or an ephemeral key when it is a Configurator.
3. Generates R-nonce.
4. Generates N, and k2.
5. Derives ke from N, M, and optionally L, and I-nonce concatenated with R-nonce.
6. Generates R-auth, the Responder's authentication tag, which is a hash of I-nonce concatenated with R-nonce, concatenated with the Initiator's public key x-coordinate ($P_I.x$), concatenated with the Responder public key x-coordinate ($P_R.x$), (optionally concatenated with the Initiator's bootstrapping key x-coordinate ($B_I.x$), when performing mutual authentication,) concatenated with the Responder's public bootstrapping key x-coordinate ($B_R.x$), concatenated with 0x0.

The Responder then sends an Authentication Response frame to the Initiator. The frame includes (see section 6.2.3):

1. The hash of the Responder's public bootstrap key (B_R).
2. Optionally includes the hash of the Initiator's public bootstrap key (B_I).
3. The Responder public protocol key (P_R).
4. R-auth wrapped with ke using AES-SIV.
5. R-nonce concatenated with I-nonce, concatenated with R-capabilities, concatenated with the wrapped R-auth is all wrapped with AES-SIV, using k2 and AAD.

Upon receiving the Responder's Authentication Response frame, the Initiator performs the following actions:

1. Unwraps the I-nonce, R-nonce, R-capabilities and R-auth wrapped with AES-SIV, using ke, with AES-SIV, using k2 and the AAD.

    If AES-SIV fails, the Initiator aborts the exchange and transitions from the Authenticating state to the Bootstrapped state, otherwise it:

    a. computes ke, and
    b. unwraps R-auth with AES-SIV, using ke.
    If AES-SIV fails, the Initiator aborts the exchange and transitions from the Authenticating state to the Bootstrapped state.
2. Verifies the Responder's authentication tag (see section 6.2.4):
    a. Generate a verification token R-auth', which is a hash of I-nonce concatenated with R-nonce, concatenated with the Initiator's public protocol key x-coordinate (PI.x), concatenated with the Responder's public protocol key x-coordinate (PR.x), (optionally concatenated with the Initiator's bootstrapping key x-coordinate (BI.x), when performing mutual authentication) concatenated with the Responder public bootstrapping key x-coordinate($B_R.x$), concatenated with 0x0.
    b. Verifies R-auth=R-auth'.
    c. If verification fails, the Initiator aborts the exchange and transitions from the Authenticating state to the Bootstrapped state.

3. Generates I-auth (see section 6.2.4), the Initiator's authentication token, which is a hash of R-nonce concatenated with I-nonce, concatenated with the Responder's public protocol key x-coordinate (P_R.x), concatenated with the Initiator's public protocol key x-coordinate (P_R.x), concatenated with the Responder public bootstrapping key x-coordinate (B_R.x), (optionally concatenated with the Initiator's bootstrapping key x-coordinate (B_I.x), when performing mutual authentication) concatenated with 0x1.
4. Responds with an Authentication Confirm frame to the Responder. The frame includes (see section 6.2.4):
   a. The hash of the Responder's public bootstrapping key (B_R).
   b. Optionally includes the hash of the Initiator's bootstrapping key (BI), when performing mutual authentication.
   c. I-auth wrapped with AES-SIV, using ke and AAD.
5. The Initiator transitions from the Authenticating state to the Authenticated state.

Upon receipt of an Authentication Confirm frame the Responder performs the following actions:

1. Unwraps I-auth with AES-SIV, using ke and AAD.
2. If AES-SIV fails, the Responder aborts the exchange.

   The Responder then transitions from the Authenticating state to the Awaiting state.
3. If the previous step succeeds, the Responder verifies the Initiator's authentication tag (see section 6.2.4):
   a. Computes I-auth', which is R-nonce concatenated with I-nonce, concatenated with the Responder's public protocol key x-coordinate (P_R.x), concatenated with the Initiator's public protocol key x-coordinate (P_I.x), concatenated with the Responder's public bootstrapping key x-coordinate (B_R.x), (optionally concatenated with the Initiator's bootstrapping key x-coordinate (B_I.x) when performing mutual authentication), concatenated with 0x1.
   b. Verifies that I-auth=I-auth':
      If verification fails, the Responder aborts the exchange. The Responder transitions from the Authenticating state to the Awaiting state.

      If verification succeeds, the Responder transitions from the Authenticating state to the Authenticated state.

At this time, the Initiator and Responder are authenticated. Their roles are set accordingly to their negotiated capabilities as Configurator and Enrollee.

7.5.3 DPP configuration exchange

A DPP configuration frame exchange takes place between an Enrollee and its Configurator. Both Enrollee and Configurator begin in the Authenticated state. The following actions take place.

The Enrollee prepares a Configuration-Request frame and sends it to the Configurator:

1. The Enrollee transmits a Configuration Request frame to the Configurator. The frame includes a series of attributes wrapped with AES-SIV, using ke and AAD.
2. The Enrollee transitions from the Authenticated state to the Provisioning state.

Upon receipt of the Configuration Request frame and if the Configurator grants the request:

1. The Configurator generates Connectors for the Enrollee. For each Connector, the Configurator:
   a. Uses the same c-sign-key/C-sign-key key pair and derived key identifier kid that identifies the network.
   b. Assigns the peer device or group-id to identify the device group with which the Enrollee can connect. The group-id may be a wildcard.
   c. Includes the Enrollee's public key, which identifies the Enrollee.
   d. Creates a label uniquely identifying the Configurator.
   e. Assigns the role of the Enrollee in the connection.

f. Computes a digital signature of the fields above with ECDSA using the signing key c-sign-key.
The Configurator generates additional configuration for the Enrollee (see sections 4.3 and 6.3.6).

2. The Configurator generates additional configuration for the Enrollee (see sections 4.3 and 6.3.6).
3. The Configurator transmits a Configuration Response frame to the Enrollee with STATUS_OK. The frame is wrapped with AES-SIV, using ke and AAD, and includes the Enrollee's E-nonce and the DPP Configuration object (see sections 4.3 and 6.3.6).
4. The Configurator transitions from the Authenticated state to the Finished state.

Upon receipt of the Configuration Request frame and if the Configurator refuses to grant the request,

1. The configurator transmits a configuration-response frame to the Enrollee with STATUS_CONFIGURE_FAILURE. The frame is wrapped with AES-SIV, using ke and AAD, and includes the Enrollee's E-nonce (see section 6.3.3).
2. The Configurator transitions from the Authenticated state to the Terminated state.

Upon receipt of the configuration-response frame with STATUS_OK, the Enrollee performs the following actions:

1. Unwraps the frame with AES-SIV using ke and AAD:
   a. If AES-SIV fails, the Enrollee may resend the Configuration Request frame.
   b. Otherwise, the Enrollee transitions from the Provisioning state to the Authenticated state.
2. Stores the DPP Configuration object.
3. Populates its cache of Connectors, which forms the list of its peers.
4. Transitions from the Provisioning state to the Provisioned state.

7.5.4 DPP network introduction exchange

This frame exchange is between the newly configured device and the other peers in the network.

1. The new peer sends out DPP Peer Discovery Request frameswhich advertise its Connector.
2. Upon receipt of a DPP Peer Discovery Request frame, any receiving peer shall perform the following actions:
   a. Decodes the Connector.
   b. Validates the kid:
   If verification fails, the receiving peer drops the message.
   c. Verifies whether the Connector is signed by the Configurator with ECDSA using C-sign-key:
   If verification fails, the receiving peer drops the message.
   d. Validates the attributes by matching the attribute objects to ensure that the peer is permitted to connect. If the attributes are group attributes, the peer evaluates each group attribute, matching the groupId with its list of groupIds, and evaluating that the netRole associated with the groupId is complimentary to its netRole in the Connector. If verification of the Connector fails, the receiving peer drops the message.
   e. Derives a PMK and PMKID by using its private key and the new peer’s public key.
   f. Responds with a DPP Peer Discovery Response frame. The message includes a hash of each public key and its own Connector.

Upon receipt of a DPP Peer Discovery Response frame, the new peer shall validate the Connector received in the DPP Peer Discovery Response frame.

At this point, each peer derives the PMK and PMKID using its private key and the other peer’s public key from the Connector.

7.5.5 Network access

1. One peer (for example, a STA) includes the DPP network introduction AKM and the PMKID in the RSN element of the Association Request frame.
2. The other peer (for example, an AP) receives the Association Request frame and validates the RSN element and responds with an Association Response frame.

Introduced peers communicate securely with WPA2 using the 4-way handshake.

8 DPP attribute, frame, and element formats

DPP messages are exchanged using 802.11 Public Action frames, with the exception of DPP Configuration request/response frames, which include a similar header format, but are exchanged using vendor specific Generic Advertisement Service (GAS) Public Action frames. The information element (IE) usage and convention for DPP Configuration request/response is the same as all other DPP frames.

In addition to DPP Protocol framing, the specification introduces the Network Introduction protocol, where peers use a Connector to establish a security association. The Network Introduction protocol elements are given in section 8.3.

8.1 DPP attributes

The DPP attributes are defined to have a common general format consisting of a 2 octet DPP Attribute ID field, a 2 octet Length field and variable-length attribute-specific information fields, as shown in Table 16. Both fields are encoded in little endian byte order.

Table 16. General format of DPP attribute

Field	Size (octets)	Value (Hexadecimal)	Description
Attribute ID	2	variable	Identifying the type of DPP attribute. The specific values are defined in Table 17.
Length	2	variable	Length of the following fields in the attribute.
Attribute body field	variable		Attribute-specific information fields.

Table 17. Attribute ID assignments

Attribute ID	Notes
0000 – 0FFF	Reserved
1000	DPP Status
1001	Initiator Bootstrapping Key Hash
1002	Responder Bootstrapping Key Hash
1003	Initiator Protocol Key
1004	Wrapped Data
1005	Initiator Nonce
1006	Initiator Capabilities
1007	Responder Nonce
1008	Responder Capabilities
1009	Responder Protocol Key
100A	Initiator Authenticating Tag
100B	Responder Authenticating Tag
100C	DPP Configuration Object, see section 6.3.6
100D	DPP Connector, see Figure 13 for an example
100E	DPP Configuration Attributes object, see section 6.3.4
100F	Bootstrapping Key
1010-1011	Reserved
1012	Finite Cyclic Group

Attribute ID	Notes
1013	Encrypted Key
1014	Enrollee Nonce
1015	Code Identifier
1016	Transaction ID
1017	Bootstrapping Info
1018	Channel
1019 - FFFF	Reserved

A DPP Device that encounters an unknown or reserved Attribute ID value in a DPP frame received without error may either drop the frame or ignore the DPP attribute and its associated fields, and parse any remaining fields for additional DPP attributes with recognizable Attribute ID values. A DPP Device that encounters a recognizable but unexpected Attribute ID value in the received DPP IE may ignore that DPP attribute.

8.1.1 DPP attribute body field definitions

8.1.1.1 DPP Status attribute

The DPP Status attribute body field is a single octet carrying a value from Table 36.

8.1.1.2 Initiator Bootstrapping Key Hash attribute and Responder Bootstrapping Key Hash attribute

The Initiator Bootstrapping Key Hash attribute and Responder Bootstrapping Key Hash attribute body fields are defined in section 6.2.2.

8.1.1.3 Initiator/ Protocol Key and Responder Protocol Key attributes

The Initiator Protocol Key and Responder Protocol Key attribute body fields are encoded as defined in section 3.2.

8.1.1.4 Wrapped Data attribute

The Wrapped Data attribute body field is defined in sections 5.6.3, 6.2 and 6.3.

8.1.1.5 Initiator Nonce, Responder Nonce, and Enrollee Nonce attributes

The Initiator Nonce, Responder Nonce, and Enrollee Nonce attribute body fields are the I-nonce, R-nonce, and E-nonce, respectively. Length is determined according to section 3.2.3.

8.1.1.6 Initiator Capabilites attribute and Responder Capabilites attribute

The DPP authentication capabilities attribute body field is a single octet bitmask that indicates the role(s) of the device as defined as shown in Figure 19. The settings for Enrollee B0 and Configurator B1 are given in Table 18.

	B0	B1	B2 B7
	Enrollee	Configurator	Reserved
Bits:	1	1	6

Figure 19. Initiator Capabilites attribute and Responder Capabilites attribute format

Table 18. Enrollee and Configurator bit settings

Enrollee (B0)	Configurator (B1)	Description
0	0	Not allowed
0	1	Device is Configurator only
1	0	Device is Enrollee only
1	1	Device is an Enrollee and Configurator (only applicable for Initator Capabilites attribute)

8.1.1.7 Initiator Authenticating Tag attribute and Responder Authenticating Tag attribute

The Initiator Authenticating Tag attribute body field is defined in section 6.2.4. The Responder Authenticating Tag attribute body field is defined in section 6.2.3.

8.1.1.8 DPP Configuration Object attribute

The DPP Configuration Object attribute body field is defined in section 6.3.4.

8.1.1.9 DPP Connector attribute

The DPP Connector attribute body field is defined in section 6.3.5.

8.1.1.10 DPP Configuration Attributes Object attribute

The DPP Configuration Attributes Object attribute body field is defined in section 6.3.6.

8.1.1.11 Bootstrapping Key attribute

The Bootstrapping Key attribute body field is defined in section 5.6.1.

8.1.1.12 Finite Cyclic Group attribute

The Finite Cyclic Group attribute body field is a two-octet integer in little endian byte order.

8.1.1.13 Encrypted Key attribute

The Encrypted Key attribute body field is defined is sections 8.2.1.7 and 8.2.1.8.

8.1.1.14 Code Identifier attribute

The Code Identifier attribute body field is defined in section 5.6.2.

8.1.1.15 Transaction ID attribute

The Transaction ID attribute body field is a one octet integer identifying a pending Peer Discovery request.

8.1.1.16 Bootstrapping Info attribute

The Bootstrapping Info attribute body field is the pkex-bootstrap-info defined in section 5.6.3.

8.1.1.17 Channel attribute

The Channel attribute body field contains two one-octet integer fields: IEEE 802.11 global operating class and channel fields (see Annex E of [2]) value pair as shown in Figure 20.

Octets:	1	1
	Operating Class	Channel

Figure 20. Channel attribute fields

8.2 DPP frames

8.2.1 DPP Public Action frames

8.2.1.1 General format

The Public Action frame format (as defined in [2]) is used to define the DPP Public Action frames in this specification. The general format of the DPP Public Action frames is shown in Table 19.

Table 19. General format of DPP Public Action frame

Field	Size (octets)	Value (Hexadecimal)	Description
Category	1	0x04	IEEE 802.11 Public Action frame usage.
Action field	1	0x09	(IEEE 802.11) vendor specific usage
OUI	3	50 6F 9A	Wi-Fi Alliance specific OUI.
OUI type	1	0x1A	Identifying the type or version of action frame. Setting to 0x1A indicates Wi-Fi Alliance DPP v1.0.
Crypto Suite	1		Indicates the crypto suite to use per section 3.2.3
DPP frame type	1		Identifying the type of DPP action frame. The specific value is defined in Table 20
Attributes	variable		A series of one or more DPP Attributes

Table 20. DPP Public Action frame type

Type	Notes
0	Authentication Request
1	Authentication Response
2	Authentication Confirm
3	Reserved
4	Reserved
5	Peer Discovery Request
6	Peer Discovery Response
7	PKEX Exchange Request
8	PKEX Exchange Response
9	PKEX Commit-Reveal Request
10	PKEX Commit-Reveal Response
11 – 255	Reserved

8.2.1.2 Authentication Request frame

The Authentication Request frame uses the DPP Public Action frame format and is transmitted by a DPP Initiator to a DPP Responder to initiate DPP authentication.

The Attributes in the Authentication Request frame are shown in Table 21.

Table 21. Attributes in the Authentication Request frame

Attribute	Required (R) / Optional (O) / Conditional (C)	Notes
Responder Bootstrapping Key Hash	R	
Initiator Bootstrapping Key Hash	R	
Initiator Protocol Key	R	
Wrapped Data	R	Ciphertext output of AES-SIV using intermediate key (k1) on Initiator nonce and Initiator Capabilities with AAD per section 6.2.
Initiator Nonce	R	This attribute is a component of the Wrapped Data
Initiator Capabilities	R	This attribute is a component of the Wrapped Data

8.2.1.3 Authentication Response frame

The Authentication Response frame uses the DPP Public Action frame format and is transmitted by a DPP Responder to a DPP Initiator to initiate a DPP authentication in response to Authentication Request frame.

The attributes included in the Authentication Response frame are shown in Table 22.

Table 22. Attributes in the Authentication Response frame

Attribute	Required (R) / Optional (O) / Conditional (C)	Notes
DPP Status	R	Error code
Responder Bootstrapping Key Hash	C	Present when DPP Status is STATUS_OK
Initiator Bootstrapping Key Hash	C	In case DPP Status is STATUS_OK, only included for mutual authentication
Responder protocol key	C	Present when DPP Status is STATUS_OK
Primary Wrapped Data	R	Ciphertext output of AES-SIV using intermediate key k2 or k1 depending on whether DPP Status is STATUS_OK or indicates an error, respectively, with AAD per section 6.2
Responder nonce	C	This is a component of the Primary Wrapped Data when DPP Status is STATUS_OK
Initiator nonce	R	This is a component of the Primary Wrapped Data
Responder capabilities	R	This is a component of the Primary Wrapped Data
Secondary Wrapped Data	C	Ciphertext output of AES-SIV using key ke on the Responder Authenticating Tag. This is a component of the Primary Wrapped Data when DPP Status is STATUS_OK, with AAD per section 6.2.
Responder Authenticating Tag	C	This is a component of the Secondary Wrapped Data

8.2.1.4 Authentication Confirm frame

The Authentication Confirm frame uses the DPP Public Action frame format and is transmitted by a DPP Initiator to a DPP Responder to confirm DPP authentication in response to an Authentication Response frame.

The attributes included in the Authentication Confirm frame are shown in Table 23.

Table 23. Attributes in the Authentication Confirm frame

Attribute	Required (R) / Optional (O) / Conditional (C)	Notes
DPP Status	R	Error code
Responder Bootstrapping Key Hash	R	
Initiator Bootstrapping Key Hash	C	Only included for mutual authentication
Wrapped Data	R	Ciphertext output of AES-SIV using key ke or k2 depending on whether DPP Status is STATUS_OK or indicates an error, respectively, with AAD per section 6.2.
Initiator Authenticating Tag	C	This is a component of Wrapped Data when DPP Status is STATUS_OK
Responder Nonce	C	This is a component of Wrapped Data when DPP Status indicates an error

8.2.1.5 Peer Discovery Request frame

The Peer Discovery Request frame uses the DPP Public Action frame format and is transmitted by a DPP Device to another DPP Device to discover peer devices.

The attributes included in the Peer Discovery Request frame are shown in Table 24.

Table 24. Attributes in the Peer Discovery Request frame

Attribute	Required (R) / Optional (O) / Conditional (C)	Notes
Transaction ID	R	Unique one octet value identifying a pending request
Connector	R	JSON encoded Connector, see section 6.3.6

8.2.1.6 Peer Discovery Response frame

The Peer Discovery Response frame uses the DPP Public Action frame format and is transmitted by a DPP Device to another DPP Device to discover peer devices in response to Peer Discovery Request frame.

The attributes included in the Peer Discovery Response frame are shown in Table 25.

Table 25. Attributes in the Peer Discovery Response frame

Attribute	Required (R) / Optional (O) / Conditional (C)	Notes
Transaction ID	R	The number identifying the transaction, copied from the Peer Discovery Request
DPP Status	R	Error code
Connector	R	JSON encoded Connector, see section 6.3.6

8.2.1.7 PKEX Exchange Request frame

The PKEX Exchange Request frame uses the DPP Public Action frame format and is transmitted by a DPP Device to another DPP Device to begin the PKEX protocol.

The attributes included in the PKEX Exchange Request frame are shown in Table 26.

Table 26. Attributes in the PKEX Exchange Request frame

Attribute	Required (R) / Optional (O) / Conditional (C)	Notes
Finite Cyclic Group	R	The group from which the public key to exchange is drawn
Code Identifier	O	An identifier of the code used for authentication, when used
Encrypted Key	R	The encrypted ephemeral key of the Initiator

8.2.1.8 PKEX Exchange Response frame

The PKEX Exchange Response frame uses the DPP Public Action frame format and is transmitted by a DPP Device to another DPP Device to respond to the initial message in the PKEX protocol.

The attributes included in the PKEX Exchange Response frame are shown in Table 27.

Table 27. Attributes in the PKEX Exchange Response frame

Attribute	Required (R) / Optional (O) / Conditional (C)	Notes
DPP Status	R	Error code
Code Identifier	O	An identifier of the code used for authentication, present if the Initiator included it in the PKEX Exchange Request frame.
Encrypted Key	C	The encrypted ephemeral key of the Responder, present if DPP Status is STATUS_OK.
Finite Cyclic Group	C	An alternate, supported group, present if DPP Status is other than STATUS_OK

8.2.1.9 PKEX Commit-Reveal Request frame

The PKEX Commit-Reveal Request frame uses the DPP Public Action frame format and is transmitted by a DPP Device to another DPP Device participating in the PKEX protocol.

The attributes included in the PKEX Commit-Reveal Request frame are shown in Table 28.

Table 28. Attributes in the PKEX Commit-Reveal Request frame

Attribute	Required (R) / Optional (O) / Conditional (C)	Notes
Wrapped Data	R	Ciphertext output of AES-SIV using key and AAD per section 5.6.1
Bootstrapping Key	R	The bootstrap key of the Initiator, this is a component of the Wrapped Data
Initiator Authenticating Tag	R	The Initiator's commitment, this is a component of the Wrapped Data
Bootstrapping Info	O	Additional information formatted per section 5.2.1.

8.2.1.10 PKEX Commit-Reveal Response frame

The PKEX Commit-Reveal Response frame uses the DPP Public Action frame format and is transmitted by a DPP Device to another DPP Device participating in the PKEX protocol.

The attributes included in the PKEX Commit-Reveal Response frame are shown in Table 29.

Table 29. Attributes in the PKEX Commit-Reveal Response frame

Attribute	Required (R) / Optional (O) / Conditional (C)	Notes
Wrapped Data	R	Ciphertext output of AES-SIV using key and AAD per section 5.6.1
Bootstrapping Key	R	The bootstrap key of the Responder, this is a component of the Wrapped Data
Responder Authenticating Tag	R	The Responder's commitment, this is a component of the Wrapped Data
Bootstrapping Info	O	Additional information formatted per section 5.2.1.

8.2.2 DPP Generic Advertisement Service (GAS) frames

8.2.2.1 General format

The DPP Configuration Request/Response frames use the Generic Advertisement Service (GAS) frame format (as defined in [2]) and include elements based on the definition of DPP Public Action frames above, with a vendor specific Advertisement protocol ID. GAS is used as it provides a fragmentation capability for the response frames, which may be required for large DPP Configuration object attributes.

The address 3 value shall be set to the broadcast address for all DPP Configuration Request/Response frames.

8.2.2.2 DPP Configuration Request frame

The DPP Configuration Request frame is transmitted by a DPP Enrollee to DPP Configurator to request configuration information.

The DPP Configuration Request is a GAS Initial Request frame with vendor specific content and is constructed using the information in Table 30.

Table 30. General format of DPP Configuration Request frame

Field	Size (octets)	Value (Hexadecimal)	Description
Category	1	0x04	IEEE 802.11 Public Action frame usage. See Table 9-47 [2]
Action field	1	0x0A	IEEE 802.11 GAS Initial Request frame usage. See Table 9-307 [2]
Dialog Token	1		Set to a value to identify the request/response transaction. See Figure 9-77 [2]
Advertisement Protocol element	3	6C 08 00	Indicates a GAS Query using a vendor specific advertisement protocol (See [2]). Element ID = 0x6C Length = 0x08 (length of all Advertisement Protocol fields) Query Response Info: 0x00 Query Response Length = 0x00 (for transmission) PAME-BI = 0x00

Field	Size (octets)	Value (Hexadecimal)	Description
Advertisement Protocol ID	7	DD 05 50 6F 9A 1A 01	Vendor-specific Element ID = 0xDD Length = 0x05 (length of OUI and type fields) OUI = 0x 50 6F 9A (Wi-Fi Alliance specific OUI) OUI Type = 0x1A Type = 0x01, denoting the DPP Configuration protocol
Query Request Length	2		Set to a nonzero value to identify the request transaction.
Query Request	variable		Contains all applicable attributes shown in Table 31

Table 31. Attributes in the DPP Configuration Request frame

Attribute	Required (R) / Optional (O) / Conditional (C)	Notes
Wrapped Data	R	Ciphertext output of AES-SIV wrapping of sub-objects.
Enrollee Nonce	R	This attribute is a component of the Wrapped Data
DPP Configuration Attribute Object	R	The JSON-encoded DPP Configuration Attribute Object attribute. This is a component of Wrapped Data.

Fragmentation

If the initial DPP Configuration Response frame (section 8.2.2.3) is a GAS Initial Response frame indicating fragmentation (non-zero comeback delay), then the DPP Configuration Response with Fragments frame, defined in Table 32, is used by the requesting STA to retrieve subsequent fragments of the configuration response. The operation of GAS fragmentation is described in section 11.25.3.2.1 of [2].

The format of the DPP Configuration Request for Fragments frame is given in Table 32.

Table 32. General format of DPP Configuration Request frame for Fragments

Field	Size (octets)	Value (Hexadecimal)	Description
Category	1	0x04	IEEE 802.11 Public Action frame usage. See Table 9-47 [2]
Action field	1	0x0C	IEEE 802.11 GAS Comeback Request usage. See Table 9-307 [2]
Dialog Token	1		Set to a value to identify the request/response transaction. It is copied from the corresponding DPP GAS Configuration Request. See Figure 9-77 in [2].

8.2.2.3 DPP Configuration Response frame

The DPP Configuration Response frame is transmitted by a DPP Configurator to a DPP Enrollee in response to DPP Configuration Request frame.

The DPP Configuration Response frame is a GAS Initial Response frame with vendor specific content and is constructed using the information in Table 33.

Table 33. General format of DPP Configuration Response frame

Field	Size (octets)	Value (Hexadecimal)	Description
Category	1	0x04	IEEE 802.11 Public Action frame usage. See Table 9-47 [2]
Action field	1	0x0B	IEEE 802.11 GAS Initial Response frame usage. See Table 9-307 [2]

Field	Size (octets)	Value (Hexadecimal)	Description
Dialog Token	1		Set to a value to identify the request/response transaction. It is copied from the DPP Configuration Request. See Figure 9077 [2]
Status Code	2		Status code as defined in [2]
GAS Comeback Delay	2	00 00 or 01 00 or larger to indicate more time is needed	This field (in little endian encoding) is set to 0 if the full response fits in a single frame or to 1 if fragmentation is needed. If the Configurator needs more than 100 ms of time to generate the response, this field indicates how long time the Enrollee should wait before sending the GAS Comeback Request frame.
Advertisement Protocol element	3	6C 08 7F	Indicates a GAS Query using a vendor specific advertisement protocol (See [2]). Element ID = 0x6C Length = 0x08 (length of all Advertisement Protocol fields) Query Response Info: 0x7F Query Response Length = 0x7F (fragments allowed) PAME-BI = 0x00
Advertisement Protocol ID	7	DD 05 50 6F 9A 1A 01	Vendor-specific Element ID = 0xDD Length = 0x05 (length of OUI and type fields) OUI = 0x 50 6F 9A (Wi-Fi Alliance specific OUI) OUI Type = 0x1A Type = 0x01 denoting the DPP Configuration protocol
Query Response Length	2		Length in octets of the Query Response field in this frame when the full response fits into a single frame. When fragmentation is needed, this is set to 0 and the Query Response is returned in subsequent GAS Comeback Response frames.
Query Response	variable		Contains all applicable attributes included in Table 34 when the full response fits into a single frame; otherwise, this field is empty in the GAS Initial Response frame and the contents is included in subsequent GAS Comeback Response frames.

Table 34. Attributes in the DPP Configuration Response frame

Attribute	Required (R) / Optional (O) / Conditional (C)	Notes
DPP Status	R	Error code
Wrapped Data	R	Ciphertext output of AES-SIV wrapping of sub-attributes.
Enrollee Nonce	R	This attribute is a component of Wrapped Data
DPP Configuration object	C	The JSON encoded DPP Configuration object. This is a component of Wrapped Data. This attribute is only present when the DPP Status attribute contains STATUS_OK.

Fragmentation

If the response exceeds the frame size, the response will be fragmented using a DPP Configuration Response for Fragments frame. The Comeback delay value is set to zero. The format of the DPP Configuration Response for Fragments frame is given in Table 35.

Table 35. General format of DPP Configuration Response frame for fragments

Field	Size (octets)	Value (Hexadecimal)	Description
Category	1	0x04	IEEE 802.11 Public Action frame usage. See Table 9-47 [2]
Action field	1	0x0D	IEEE 802.11 GAS Comeback Response frame usage. See Table 9-307 [2]

Field	Size (octets)	Value (Hexadecimal)	Description
Dialog Token	1		Set to a value to identify the request/response transaction. It is copied from the DPP Configuration Request frame. See Figure 8-77 [2]
Status Code	2		Status code as defined in[2]
GAS Query Response Fragment ID	1		GAS fragment number. See section 9.4.1.34 [2]
GAS Comeback Delay	2	00 00	
Advertisement Protocol element	3	6C 08 7F	Indicates a GAS Query using a vendor specific advertisement protocol (See [2]). Element ID = 0x6C Length = 0x08 (length of all Advertisement Protocol fields) Query Response Info: 0x7F Query Response Length = 0x7F (fragments allowed) PAME-BI = 0x00
Advertisement Protocol ID	7	DD 05 50 6F 9A 1A 01	Vendor-specific Element ID = 0xDD Length = 0x05 (length of OUI and type fields) OUI = 0x 50 6F 9A (Wi-Fi Alliance specific OUI) OUI Type = 0x1A Type = 0x01, denoting the DPP Configuration protocol
Query Response Length	2		Length in octets of the Query Response field in the frame.
Query Response	variable		Contains all applicable attributes included in Table 34 fragmented over two or more GAS Comeback Response frames.

8.3 DPP status and error codes

The DPP Authentication, Configuration and Network Introduction protocols as well as PKEX use DPP Status fields to represent status as well as communicate errors. The possible values for DPP Status in the DPP protocols and PKEX are defined in Table 36.

Table 36. DPP status and error codes

Status or Error	Value	Meaning
STATUS_OK	0	No errors or abnormal behavior
STATUS_NOT_COMPATIBLE	1	The DPP Initiator and Responder have incompatible capabilities
STATUS_AUTH_FAILURE	2	Authentication failed
STATUS_BAD_CODE	3	The code used in PKEX is bad
STATUS_BAD_GROUP	4	An unsupported group was offered
STATUS_CONFIGURE_FAILURE	5	Configurator refused to configure Enrollee
STATUS_RESPONSE_PENDING	6	Responder will reply later
STATUS_INVALID_CONNECTOR	7	Received Connector is invalid for some reason. The sending device needs to be reconfigured.
STATUS_NO_MATCH	8	Received Connector is verified and valid but no matching Connector could be found. The receiving device needs to be reconfigured.

8.4 Network Introduction protocol elements

8.4.1 Overview

When peers use Connectors to establish a security association, they shall use the AKM defined in section 8.4.2 and follow the security procedures described in Clause 12.6 of [2].

8.4.2 Network Introduction protocol AKM suite

The Network Introduction protocol AKM Suite Selector is included in the RSN element as defined in Clause 9.4.2.25.3 of [2]. PMF shall be enabled for all associations using the Network Introduction protocol AKM.

The Network Introduction protocol AKM Suite Selector is defined in Table 37. References in the table point to clauses in [2]. A peer using this AKM shall use PRF from [2] with the hash algorithm given in Table 3 for PTK derivation, which is dependent on the length of the prime.

The length of the hash algorithm output shall be used as the length of PMK. The Nonce length in Table 3 shall be used as the length of KCK and EAPOL-Key MIC. The length of KEK shall be 128 if the Nonce length is 128; otherwise, the length of KEK shall be 256. The NIST AES key wrap shall be used as the EAPOL-Key encryption algorithm. The Integrity Algorithm is HMAC with the hash algorithm from Table 3. The Key Descriptor Version shall be set to zero on all transmitted EAPOL-Key frames.

Table 37. AKM Suite selector for Network Introduction protocol

OUI	Suite Type	Meaning		
		Authentication Type	Key Management Type	Key Derivation Type
50 6F 9A	2	Authentication with Hash Algorithm given in Table 3	RSNA key management as defined in section 12.7 [2] or using PMK caching defined in section 12.6.10.3 [2] – Cached PMKSAs and RSNA key management	Defined in section 12.7.1.7.2 [2] using Hash Algorithm given in Table 3 as Hash.

9 DPP configuration backup and restore

NOTE: THIS FEATURE HAS NOT BEEN TESTED IN THE DEVICE PROVISIONING PROTOCOL (DPP) CERTIFICATION PROGRAM.

9.1 Overview

This section specifies the syntax for DPP Configurator's configuration information backup and restore.

Section 9.2 specifies the DPPAsymmetricKeyPackage.

Section 9.3 specifies the DPPEnvelopedData, which is the encrypted form of the DPPAsymmetricKeyPackage.

Section 9.4 and section 9.5 specifies the procedures for the Configurator's configuration backup and restore.

9.2 DPP AsymmetricKeyPackage

The syntax for the DPPAsymmetricKeyPackage is a profiled version of AsymmetricKeyPackage (RFC 5958 [29]).

```
DPPAsymmetricKeyPackage ::= AsymmetricKeyPackage

AsymmetricKeyPackage ::= SEQUENCE SIZE (1..MAX) OF OneAsymmetricKey

OneAsymmetricKey ::= SEQUENCE {
       version             Version,
       privateKeyAlgorithm    PrivateKeyAlgorithmIdentifier,
       privateKey          PrivateKey,
       attributes                  [0] Attributes OPTIONAL,
       ...,
       [[2: publicKey              [1] BIT STRING OPTIONAL ]],
       …
}

Version ::= INTEGER { v1(0), v2(1) } (v1, ..., v2)

PrivateKeyAlgorithmIdentifier ::= AlgorithmIdentifier

PrivateKey ::= OCTET STRING  -- Contains DER encoding of ECPrivateKey

Attributes ::= SET OF Attribute { { OneAsymmetricParameters } }
```

The fields in DPPAsymmetricKeyPackage are as follows:

```
version is set to v2(1).
privateKeyAlgorithm is set to ECDSA or ECDH.
privateKey is an OCTET STRING that contains the ECPrivateKey SEQUENCE as specified in [32]:

ECPrivateKey ::= SEQUENCE {
     version                  INTEGER { ecPrivkeyVer1(1) } (ecPrivkeyVer1),
     privateKey               OCTET STRING,
     parameters[0]            ECParameters {{ NamedCurve }},
                      publicKey[1]           BIT STRING OPTIONAL}
```

Attributes shall contain the following attribute:

```
OneAsymmetricKeyAttributes ATTRIBUTE ::= {
       aa-DPPConfigurationParameters,
       ... -- For local profiles
```

```
            }

      aa-DPPConfigurationParameters ATTRIBUTE ::=
            { TYPE DPPConfigurationParameters IDENTIFIED BY id-DPPConfigParams }

      wfaProject OBJECT IDENTIFIER ::= { 1 3 6 1 4 1 40808 1 }
      wfaDpp OBJECT IDENTIFIER ::= { wfaProject 2

      id-DPPConfigParams OBJECT IDENTIFER ::= { wfaDpp 1}

      DPPConfigurationParams ::= SEQUENCE {
              configurationTemplate   UTF8String,
              connectorTemplate       UTF8String OPTIONAL
              }
}
```

Specifically:

- configurationTemplate corresponds to a JSON configuration object as defined in Table 14 minus the fields C-sign-key and signedConnector.
- connectorTemplate corresponds to a JSON object as defined in Table 13 minus the netAccessKey field.
- connectorTemplate is an optional field of the Configuration sequence.

Attributes shall contain the following sequences:

```
ConfigurationParams ::= SEQUENCE {
      ConfigurationTemplate   OCTET STRING,
    [ConnectorTemplate        OCTET STRING  OPTIONAL]
}
```

Specifically:

- ConfigurationTemplate corresponds to a JSON configuration object as defined in Table 14 minus the fields C-sign-key and signedConnector.
- ConnectorTemplate corresponds to a JSON object as defined in Table 13 minus the netAccessKey field.
- ConnectorTemplate is an optional field of the Configuration sequence.

9.3 DPPEnvelopedData

The syntax for the DPPEnvelopedData is a profiled version of EnvelopedData as defined in [29].

In RFC 5652 [28], an enveloped-data content type consists of encrypted content of any type and an encrypted content-encryption key for one or more recipients. The combination of the encrypted content and one encrypted content-encryption key is called a "digital envelope". Any type of content can be enveloped for an arbitrary number of recipients.

DPPEnvelopedData encrypts the DPPAsymmetricKeyPackage with a content-encryption key derived from a password.

```
DPPEnvelopedData ::= EnvelopedData

EnvelopedData ::= SEQUENCE {
       version                      CMSVersion,
       originatorInfo         [0]   IMPLICIT OriginatorInfo OPTIONAL,
       recipientInfos               DPPRecipientInfos,
       encryptedContentInfo         EncryptedContentInfo,
       unprotectedAttrs [1] IMPLICIT  UnprotectedAttributes OPTIONAL}

RecipientInfos ::= SET SIZE (1..MAX) OF RecipientInfo
```

Specifically:

```
version shall be assigned to 3.
originatorInfo is omitted.
recipientInfo is described below.
recipientInfos shall always contain a single RecipientInfo as described below.
encryptedContentInfo shall always contain the encrypted DPPAsymmetricKeyPackage as described
below. There is only one recipientInfo in the sequence.
unprotectedAttrs shall always be absent.

The EncryptedContentInfo contains the encrypted DPPAsymmetricKeyPackage.

EncryptedContentInfo ::= SEQUENCE {
        contentType                 ContentType,
        contentEncryptionAlgorithm     ContentEncryptionAlgorithmIdentifier,
        encryptedContent [0] IMPLICIT  EncryptedContent OPTIONAL  }

EncryptedContent ::= OCTET STRING
id-DPPAsymmetricKeyPackage OBJECT IDENTIFIER ::= { wfaDpp 2 }

id-DPPaesSiv OBJECT IDENTIFIER :== { wfaDpp 3 }


Specifically:
contentType shall always be id-DPPAsymetricKeyPackage, indicating that the encrypted content
contains a DPPAsymmetricKeyPackage.
contentEncryptionAlgorithm shall always indicate that the content was encrypted using AES-SIV
(id-DPPaesSiv).
encryptedContent shall always be present, and the OCTET STRING shall contain the result of
encrypting the DPPAsymmetricKeyPackage with AES-SIV and the key derived from a password as
described below.

The RecipientInfo shall always use the pwri CHOICE as described in RFC 5652.

RecipientInfo ::= CHOICE {
        ktri   KeyTransRecipient,
        kari [1]  KeyAgreeRecipientInfo,
        kekri [2]KEKRecipientInfo,
        pwri [3]  PasswordRecipientinfo,
        ori [4]   OtherRecipientInfo }

PasswordRecipientInfo ::= SEQUENCE {
        version          CMSVersion,
        keyDerivationAlgorithm [0]  KeyDerivationAlgorithmIdentifier OPTIONAL,
        keyEncryptionAlgorithm   KeyEncryptionAlgorithmIdentifier,
        encryptedKey        EncryptedKey }

Specifically:
version shall always be set to 0.
keyDerivationAlgorithm shall always be id-PBKDF2 as specified in RFC 8018, and the parameters
shall always contain the PBKDF2-params SEQUENCE as described below
keyEncryptionAlgorithm shall always indicate that the content was encrypted using AES-SIV
(id-DPPaesSiv).
encryptedKey is the result of encrypting the fresh content-encryption key with AES-SIV using
the key-encryption key derived from the password.

PBKDF2, as specified in RFC 8018, shall always be used as key derivation algorithm, and the
parameters below shall always be used to derive the AES-SIV key-encryption key from the user-
supplied password.

id-PBKDF2 OBJECT IDENTIFIER ::= { 1 2 840 113549 1 5 12 }

PBKDF2-params ::= SEQUENCE {
```

```
        salt   CHOICE {
specified OCTET STRING,
                otherSource AlgorithmIdentifier },
        iterationCount   INTEGER (1..MAX),
        keyLength   INTEGER (1..MAX),
        prf      AlgorithmIdentifier}

Specifically:

salt shall be set to a random or pseudorandom 64 octet value.
iterationCount shall be set to 1000.
keyLength shall be set as indicated in Table 3.
prf shall be set id-sha256 as specified in RFC 5912, which identifies the SHA-256 algorithm.

Note that Salt, c, and dkLen parameters are obtained from the PBKDF2-params in the
keyDerivationAlgorithm field of the RecipientInfo.
```

9.3.1 DPPAsymmetricKeyPackage encryption

To construct the DPPEnvelopedData from a DPPAsymmetricKeyPackage, the encryption procedure is composed of the following steps:

1. Select a salt S and set the iteration count c to 1000.
2. Select the length in octets, dkLen, for the derived key for the underlying encryption scheme.
3. Apply the selected key derivation function to the password P, the salt S, and the iteration count c to produce a derived key DK of length dkLen octets: DK = PBKDF2 (P, S, c, dkLen).
4. Encrypt the DPPAsymmetricKeyPackage with the underlying encryption scheme under the derived key DK to produce the DPPEnvelopedData. This step may involve selection of parameters such as an initialization vector and padding, depending on the underlying scheme.

9.3.2 DPPEnvelopedData decryption

The decryption operation of the DPPEnvelopedData consists of the following steps:

1. Obtain the salt S for the operation.
2. Obtain the iteration count c for the key derivation function.
3. Obtain the key length in octets, dkLen, for the derived key for the underlying encryption scheme.
4. Apply the selected key derivation function to the password P, the salt S, and the iteration count c to produce a derived key DK of length dkLen octet: DK = PKBDF2 (P, S, c, dkLen).
5. Decrypt the DPPEnvelopedData with the underlying encryption schema under the derived key DK to recover the DPPAsymmetricKeyPackage. If the decryption function outputs "decryption error," then output "decryption error" and stop.

9.4 DPP configuration backup

The DPP digital envelope should be stored at a designated uniform resource identifier in the DPP network, for example, wireless drive or AP, or onto a removable media. The methods to store and retrieve the DPP Digital Envelop, including the uniform resource identifier, are vendor specific. The owner / manager of the network is responsible for managing the password to protect the content-encryption key.

9.5 DPP configuration restore

To restore a Configurator’s configuration on a new device, the DPP digital envelope is retrieved from a designated uniform resource identifier in the DPP network, for example, wireless drive or the AP, or removable media. The owner/manager of

the network uses the password method for deriving the content-encryption key to decrypt the DPP digital envelope on the new device to restore/replace the old Configurator's configuration on a new device. Upon decryption, the information in the DPP AsymmetricKeyPackage, including the (signature, verification) key pairs, can be used by the restored Configurator to manage the network and enroll new devices.

A recipient opens the digital envelope by decrypting one of the encrypted content-encryption key and then decrypting the encrypted content with the recovered content-encryption key.

9.6 Enabling multiple Configurators in DPP

Enabling multiple Configurators in DPP consist in multiple Configurators sharing the (signature, verification) key pair, as well as additional information, which is to be used to allow a new Configurator to gauge a consistent view of the network.

The procedure to backup and restore in DPP shall be used to enable multiple Configurators. Specifically, the current Configurator prepares a DPP digital envelop as specified in section 9.4. The DPP digital envelop shall be stored at a designated uniform resource identifier.

The restore procedure described in section 9.5 shall be used to enable multiple Configurators, by restoring the configuration on one or multiple new devices which the owner / manager of the network wants to use as additional Configurators.

Appendix A (Informative) Test vectors

A.1 Test vectors for DPP Authentication using P-256 for mutual authentication

This section contains test vectors for the DPP Authentication protocol using curve P-256 and mutual authentication.

```
Curve used: NIST P-256
INITIATOR VALUES
Initiator is: Configurator

I-capabilities (1 octets):
02

Private bootstrapping key (32 octets):
15b2a83c 5a0a38b6 1f2aa820 0ee4994b 8afdc01c 58507d10 d0a38f7e edf051bb

Public  bootstrapping key
x (32 octets):
88b37ed9 1938b519 7097808a 62448476 17892046 d93b9501 afd48fa0 f148dfde

y (31 octets):
f73b6991 287884a9 c9a33f8e 0691f14d 44b59811 e9d8242d 010270b0 d33ec0

Private protocol key (32 octets):
a87de9af bb406c96 e5f79a3d f895ecac 3ad406f9 5da66314 c8cb3165 e0c61783

Public  protocol key
x (32 octets):
50a532ae 2a072072 76418d2f a630295d 45569be4 25aa634f 02014d00 a7d1f61a

y (32 octets):
e14f35a5 a858bcca d90d126c 46594c49 ef82655e 78888e15 a32d916a c2172491

DER encoded ASN.1 (input to the SHA256 hash) (59 octets):
30393013 06072a86 48ce3d02 0106082a 8648ce3d 03010703 22000288 b37ed919
38b51970 97808a62 44847617 892046d9 3b9501af d48fa0f1 48dfde

DER encoded ASN.1 base64 for use in the QR code (80 octets):
MDkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDIgACiLN+2Rk4tRlwl4CKYkSEdheJIEbZO5UBr9SPoPFI394=

I-nonce (16 octets):
13f4602a 16daeb69 712263b9 c46cba31
```

RESPONDER VALUES

Responder is: Enrollee that wants to do mutual authentication

```
R-capabilities (1 octets):
01

Private bootstrapping key (32 octets):
54ce181a 98525f21 7216f59b 245f60e9 df30ac7f 6b26c939 418cfc3c 42d1afa0

Public  bootstrapping key
x (32 octets):
09c585a9 1b4df9fd 25a04520 1885c39c c5cfae39 7ddaeda9 57dec57f a0e3503f
```

```
y (32 octets):
52bf0596 8198a2f9 2883e96a 386d7675 79883302 dbf29210 5c90a436 94c2fd5c

Private protocol key (32 octets):
f798ed2e 19286f6a 6efe210b 1863badb 99af2a14 b497634d bfd2a973 94fb5aa5

Public  protocol key
x (32 octets):
5e3fb357 6884887f 17c3203d 8a3a6c2f ac722ef0 e2201b61 ac73bc65 5c709a90

y (32 octets):
2d4b0306 69fb9eff 8b0a79fa 7c1a172a c2a92c62 6256963f 9274dc90 682c81e5

DER encoded ASN.1 (input to the SHA256 hash) (59 octets):
30393013 06072a86 48ce3d02 0106082a 8648ce3d 03010703 22000209 c585a91b
4df9fd25 a0452018 85c39cc5 cfae397d daeda957 dec57fa0 e3503f

DER encoded ASN.1 base64 for use in the QR code (80 octets):
MDkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDIgACCcWFqRtN+f0loEUgGIXDnMXPrjl92u2pV97Ff6DjUD8=

R-nonce (16 octets):
3d0cfb01 1ca916d7 96f7029f f0b43393
```

Shared secrets

```
Secret M
x (32 octets):
dde28781 17d69745 be4f916a 2dd14269 d783d1d7 88c603bb 8746beab bd1dbbbc

y (32 octets):
277f4abe 025c5759 5fbed85b a90df129 b7610364 4b2611d9 66edc23d 2650961d


Secret k1 (32 octets):
3d832a02 ed6d7fc1 dc96d2ec eab738cf 01c0028e b256be33 d5a21a72 0bfcf949

Secret N
x (32 octets):
92118478 b75c21c2 c59340c8 42b5bce5 60a535f6 0bc37a75 fe390d73 8c58d8e8

y (32 octets):
ed71f714 46560207 01fe281f a851fa5a e7840a71 bce8a266 3e3deaff 50b6985d


Secret k2 (32 octets):
ca08bdee ef838ddf 897a5f01 f20bb93d c5a895cb 86788ca8 c00a7664 899bc310

Secret L
x (32 octets):
fb737234 c973cc3a 36e64e51 70a32f12 089d198c 73c2fd85 a53d0b28 2530fd02

y (32 octets):
9876c937 b642cce8 f604bef0 c24ce0da d2c27867 08a7575a f940a58c fe8ca3ce

Secret ke (32 octets):
b6db6552 6c9a0174 c3bed56f 7e614f3a 656233c0 78693249 ac351642 5127e5d5

I-auth (32 octets):
d34944bb 4b1f05ca ebda762c 6e4ae034 c819ec2f 62a57dcf ade24738 76e007b2
```

```
R-auth (32 octets):
a725abe6 dc66ccf3 aa3d6d61 a19932fc bb0799ed 09ff78e5 bc6d4ea5 ef8e8670

XXXX Unknown frame duration (msec)
YYYY Unknown fragment/Sequence number
```

DPP Auth Request frame from 9f:86:64:c1:b8:77 to 1a:c4:59:c4:0d:64

```
d000XXXX 1ac459c4 0d649f86 64c1b877 1ac459c4 0d64YYYY 0409506f 9a1a0100
02102000 922ddd7a 3ed69f46 125d772b be6017cd 4e03870d c014509e 38b54628
e157a87d 01102000 5d467a09 760292fc 15d31792 b0a5b050 db8bf6ad 807d71b2
d93f4d1c 2e65d881 03104000 50a532ae 2a072072 76418d2f a630295d 45569be4
25aa634f 02014d00 a7d1f61a e14f35a5 a858bcca d90d126c 46594c49 ef82655e
78888e15 a32d916a c2172491 18100200 51010410 2900868f 478fc599 ac3fa815
2b975eff 8be4e71b 189dbefb c3185b1d 7f3864e8 96f913cb a3d96013 26f278

Fields of the Authentication Request message (with tag and length part omitted)
Auth req  field1, Responder Bootstrap Hash (32 octets):
922ddd7a 3ed69f46 125d772b be6017cd 4e03870d c014509e 38b54628 e157a87d

Auth req  field2, Initiator Bootstrap Hash (32 octets):
5d467a09 760292fc 15d31792 b0a5b050 db8bf6ad 807d71b2 d93f4d1c 2e65d881

Auth req  field3, Initiator Protocol Key   (64 octets):
50a532ae 2a072072 76418d2f a630295d 45569be4 25aa634f 02014d00 a7d1f61a
e14f35a5 a858bcca d90d126c 46594c49 ef82655e 78888e15 a32d916a c2172491

Auth req  field4, Channel                  (2 octets):
5101

Auth req  field5, Data wrapped with k1     (41 octets):
868f478f c599ac3f a8152b97 5eff8be4 e71b189d befbc318 5b1d7f38 64e896f9
13cba3d9 601326f2 78
```

DPP Auth Response frame from 1a:c4:59:c4:0d:64 to 9f:86:64:c1:b8:77

```
d000XXXX 9f8664c1 b8771ac4 59c40d64 9f8664c1 b877YYYY 0409506f 9a1a0101
00100100 00021020 00922ddd 7a3ed69f 46125d77 2bbe6017 cd4e0387 0dc01450
9e38b546 28e157a8 7d011020 005d467a 09760292 fc15d317 92b0a5b0 50db8bf6
ad807d71 b2d93f4d 1c2e65d8 81091040 005e3fb3 57688488 7f17c320 3d8a3a6c
2fac722e f0e2201b 61ac73bc 655c709a 902d4b03 0669fb9e ff8b0a79 fa7c1a17
2ac2a92c 62625696 3f9274dc 90682c81 e5041075 0061058b 7f101548 071c1702
26c23634 3e3f4e7c 9e2491e1 14ed461b a5b6e869 65f7f399 e47edbe9 3ba3f666
1db68a49 3985d421 d1e8a3fc b03d1c93 987551ca 844882af a74c56d6 027ec748
b7731f4c 051b509c 2a9e1932 fa39b0e9 671f0949 61a191d0 92333d69 50e6e60f
bad52857 7ca306bb 7b37

Fields of the Authentication Response message (with tag and length part omitted)
Auth resp field0, DPP_Status               (1 octets):
00

Auth resp field1, Responder Bootstrap Hash (32 octets):
922ddd7a 3ed69f46 125d772b be6017cd 4e03870d c014509e 38b54628 e157a87d

Auth resp field2, Initiator Bootstrap Hash (32 octets):
```

```
5d467a09 760292fc 15d31792 b0a5b050 db8bf6ad 807d71b2 d93f4d1c 2e65d881

Auth resp field3, Responder Protocol Key   (64 octets):
5e3fb357 6884887f 17c3203d 8a3a6c2f ac722ef0 e2201b61 ac73bc65 5c709a90
2d4b0306 69fb9eff 8b0a79fa 7c1a172a c2a92c62 6256963f 9274dc90 682c81e5

Auth resp field4, Data wrapped with k2     (117 octets):
61058b7f 10154807 1c170226 c236343e 3f4e7c9e 2491e114 ed461ba5 b6e86965
f7f399e4 7edbe93b a3f6661d b68a4939 85d421d1 e8a3fcb0 3d1c9398 7551ca84
4882afa7 4c56d602 7ec748b7 731f4c05 1b509c2a 9e1932fa 39b0e967 1f094961
a191d092 333d6950 e6e60fba d528577c a306bb7b 37
```

DPP Auth Confirm frame from 9f:86:64:c1:b8:77 to 1a:c4:59:c4:0d:64

```
d000XXXX 1ac459c4 0d649f86 64c1b877 1ac459c4 0d64YYYY 0409506f 9a1a0102
00100100 00021020 00922ddd 7a3ed69f 46125d77 2bbe6017 cd4e0387 0dc01450
9e38b546 28e157a8 7d011020 005d467a 09760292 fc15d317 92b0a5b0 50db8bf6
ad807d71 b2d93f4d 1c2e65d8 81041034 00cb4d07 49561a33 0db9f76b fdef6fcb
7d7949be d87eb685 63ac4dd1 ab8bba0f d9a525e2 8e5253db 040195ef 69aec9f0
f1e52426 38

Fields of the Authentication Confirm frame (with tag and length part omitted)
Auth conf field0, DPP_Status               (1 octets):
00

Auth conf field1, Responder Bootstrap Hash (32 octets):
922ddd7a 3ed69f46 125d772b be6017cd 4e03870d c014509e 38b54628 e157a87d

Auth conf field2, Initiator Bootstrap Hash (32 octets):
5d467a09 760292fc 15d31792 b0a5b050 db8bf6ad 807d71b2 d93f4d1c 2e65d881

Auth conf field3, Data wrapped with ke     (52 octets):
cb4d0749 561a330d b9f76bfd ef6fcb7d 7949bed8 7eb68563 ac4dd1ab 8bba0fd9
a525e28e 5253db04 0195ef69 aec9f0f1 e5242638
```

A.2 Test vectors for DPP Authentication using P-256 for Responder-only authentication

This section contains test vectors for the DPP Authentication protocol using curve P-256 and only the Responder being authenticated.

```
Curve used: NIST P-256
INITIATOR VALUES
Initiator is: Configurator

I-capabilities (1 octets):
02

Private protocol key (32 octets):
a87de9af bb406c96 e5f79a3d f895ecac 3ad406f9 5da66314 c8cb3165 e0c61783

Public  protocol key
x (32 octets):
50a532ae 2a072072 76418d2f a630295d 45569be4 25aa634f 02014d00 a7d1f61a

y (32 octets):
e14f35a5 a858bcca d90d126c 46594c49 ef82655e 78888e15 a32d916a c2172491
```

```
I-nonce (16 octets):
13f4602a 16daeb69 712263b9 c46cba31
```

RESPONDER VALUES

```
Responder is: Enrollee that does not want to do mutual authentication)

R-capabilities (1 octets):
01

Private bootstrapping key (32 octets):
54ce181a 98525f21 7216f59b 245f60e9 df30ac7f 6b26c939 418cfc3c 42d1afa0

Public  bootstrapping key
x (32 octets):
09c585a9 1b4df9fd 25a04520 1885c39c c5cfae39 7ddaeda9 57dec57f a0e3503f

y (32 octets):
52bf0596 8198a2f9 2883e96a 386d7675 79883302 dbf29210 5c90a436 94c2fd5c

Private protocol key (32 octets):
f798ed2e 19286f6a 6efe210b 1863badb 99af2a14 b497634d bfd2a973 94fb5aa5

Public  protocol key
x (32 octets):
5e3fb357 6884887f 17c3203d 8a3a6c2f ac722ef0 e2201b61 ac73bc65 5c709a90

y (32 octets):
2d4b0306 69fb9eff 8b0a79fa 7c1a172a c2a92c62 6256963f 9274dc90 682c81e5

DER encoded ASN.1 (input to the SHA256 hash) (59 octets):
30393013 06072a86 48ce3d02 0106082a 8648ce3d 03010703 22000209 c585a91b
4df9fd25 a0452018 85c39cc5 cfae397d daeda957 dec57fa0 e3503f

DER encoded ASN.1 base64 for use in the QR code (80 octets):
MDkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDIgACCcWFqRtN+f0loEUgGIXDnMXPrjl92u2pV97Ff6DjUD8=

R-nonce (16 octets):
3d0cfb01 1ca916d7 96f7029f f0b43393
```

Shared secrets

```
Secret M
x (32 octets):
dde28781 17d69745 be4f916a 2dd14269 d783d1d7 88c603bb 8746beab bd1dbbbc

y (32 octets):
277f4abe 025c5759 5fbed85b a90df129 b7610364 4b2611d9 66edc23d 2650961d

Secret k1 (32 octets):
3d832a02 ed6d7fc1 dc96d2ec eab738cf 01c0028e b256be33 d5a21a72 0bfcf949

Secret N
x (32 octets):
92118478 b75c21c2 c59340c8 42b5bce5 60a535f6 0bc37a75 fe390d73 8c58d8e8

y (32 octets):
ed71f714 46560207 01fe281f a851fa5a e7840a71 bce8a266 3e3deaff 50b6985d
```

```
Secret k2 (32 octets):
ca08bdee ef838ddf 897a5f01 f20bb93d c5a895cb 86788ca8 c00a7664 899bc310

Secret L
x (0 octets):

y (0 octets):

Secret ke (32 octets):
c8882a8a b30c8784 67822534 138c704e de0ab1e8 73fe03b6 01a79084 63fec87a

I-auth (32 octets):
787d1189 b526448d 2901e7f6 c22775ce 514fce52 fc886c1e 924f2fbb 8d97b210

R-auth (32 octets):
43509ef7 137d8c2f be66d802 ae09dedd 94d41b8c bfafb495 4782014f f4a3f91c

XXXX Unknown frame duration (msec)
YYYY Unknown fragment/Sequence number
```

DPP Auth Request frame from 9f:86:64:c1:b8:77 to 1a:c4:59:c4:0d:64

```
d000XXXX 1ac459c4 0d649f86 64c1b877 1ac459c4 0d64YYYY 0409506f 9a1a0100
02102000 922ddd7a 3ed69f46 125d772b be6017cd 4e03870d c014509e 38b54628
e157a87d 01102000 5d467a09 760292fc 15d31792 b0a5b050 db8bf6ad 807d71b2
d93f4d1c 2e65d881 03104000 50a532ae 2a072072 76418d2f a630295d 45569be4
25aa634f 02014d00 a7d1f61a e14f35a5 a858bcca d90d126c 46594c49 ef82655e
78888e15 a32d916a c2172491 18100200 51010410 2900868f 478fc599 ac3fa815
2b975eff 8be4e71b 189dbefb c3185b1d 7f3864e8 96f913cb a3d96013 26f278

Fields of the Authentication Request message (with tag and length part omitted)
Auth req  field1, Responder Bootstrap Hash (32 octets):
922ddd7a 3ed69f46 125d772b be6017cd 4e03870d c014509e 38b54628 e157a87d

Auth req  field2, Initiator Bootstrap Hash (32 octets):
5d467a09 760292fc 15d31792 b0a5b050 db8bf6ad 807d71b2 d93f4d1c 2e65d881

Auth req  field3, Initiator Protocol Key   (64 octets):
50a532ae 2a072072 76418d2f a630295d 45569be4 25aa634f 02014d00 a7d1f61a
e14f35a5 a858bcca d90d126c 46594c49 ef82655e 78888e15 a32d916a c2172491

Auth req  field4, Channel                  (2 octets):
5101

Auth req  field5, Data wrapped with k1     (41 octets):
868f478f c599ac3f a8152b97 5eff8be4 e71b189d befbc318 5b1d7f38 64e896f9
13cba3d9 601326f2 78
```

DPP Auth Response frame from 1a:c4:59:c4:0d:64 to 9f:86:64:c1:b8:77

```
d000XXXX 9f8664c1 b8771ac4 59c40d64 9f8664c1 b877YYYY 0409506f 9a1a0101
00100100 00021020 00922ddd 7a3ed69f 46125d77 2bbe6017 cd4e0387 0dc01450
9e38b546 28e157a8 7d091040 005e3fb3 57688488 7f17c320 3d8a3a6c 2fac722e
f0e2201b 61ac73bc 655c709a 902d4b03 0669fb9e ff8b0a79 fa7c1a17 2ac2a92c
62625696 3f9274dc 90682c81 e5041075 00da553c df80da3e 27054c5e 1f809ac3
```

```
03c63948 b9bb5690 ad12f357 d75dfbc3 62cbae89 e472dd68 51925534 024310af
f5ae4038 31e98a7e fc7deb95 16164329 c227039a e73c5091 47d156ae 085f56c2
42bf7dec c1f3b68d 81697c61 97453cb6 faff7b06 2f786107 3148052d b539895b
c6583d08 b4aa

Fields of the Authentication Response message (with tag and length part omitted)
Auth resp field0, DPP_Status              (1 octets):
00

Auth resp field1, Responder Bootstrap Hash (32 octets):
922ddd7a 3ed69f46 125d772b be6017cd 4e03870d c014509e 38b54628 e157a87d

Auth resp field3, Responder Protocol Key   (64 octets):
5e3fb357 6884887f 17c3203d 8a3a6c2f ac722ef0 e2201b61 ac73bc65 5c709a90
2d4b0306 69fb9eff 8b0a79fa 7c1a172a c2a92c62 6256963f 9274dc90 682c81e5

Auth resp field4, Data wrapped with k2     (117 octets):
da553cdf 80da3e27 054c5e1f 809ac303 c63948b9 bb5690ad 12f357d7 5dfbc362
cbae89e4 72dd6851 92553402 4310aff5 ae403831 e98a7efc 7deb9516 164329c2
27039ae7 3c509147 d156ae08 5f56c242 bf7decc1 f3b68d81 697c6197 453cb6fa
ff7b062f 78610731 48052db5 39895bc6 583d08b4 aa
```

DPP Auth Confirm frame from 9f:86:64:c1:b8:77 to 1a:c4:59:c4:0d:64

```
d000XXXX 1ac459c4 0d649f86 64c1b877 1ac459c4 0d64YYYY 0409506f 9a1a0102
00100100 00021020 00922ddd 7a3ed69f 46125d77 2bbe6017 cd4e0387 0dc01450
9e38b546 28e157a8 7d041034 0054e07e 62c74526 dfd97e02 9dc781e0 771e573e
bc73c942 27b5de83 50fc6a19 74b40f54 c9fe1a1c 9288a91f ce4ee6c1 f2ff0697
41

Fields of the Authentication Confirm frame (with tag and length part omitted)
Auth conf field0, DPP_Status              (1 octets):
00

Auth conf field1, Responder Bootstrap Hash (32 octets):
922ddd7a 3ed69f46 125d772b be6017cd 4e03870d c014509e 38b54628 e157a87d

Auth conf field3, Data wrapped with ke     (52 octets):
54e07e62 c74526df d97e029d c781e077 1e573ebc 73c94227 b5de8350 fc6a1974
b40f54c9 fe1a1c92 88a91fce 4ee6c1f2 ff069741
```

A.3 Test vectors for DPP Authentication using P-384 for mutual authentication

This section contains test vectors for the DPP Authentication protocol using curve P-384 and mutual authentication.

```
Curve used: NIST P-384
INITIATOR VALUES
Initiator is: Configurator

I-capabilities (1 octets):
02

Private bootstrapping key (48 octets):
40428496 d8bbb1a6 8c8bf2b6 bd1ec803 9859e79d 9c3de9dc 675154c2 c0a37c3f
0c7eed79 bf212e0d 48fcca28 42dade8c

Public  bootstrapping key
```

```
x (48 octets):
b0d7d4ec bd61c96c f246a214 5cd797dc 1d0fd3b3 539b3e8e 2ace411d 4ed736b9
5f29c73d 515cbc1c 8e9184d1 26361f9b

y (48 octets):
a5edf15b 7e5336ab 119e0507 ebab2dae 110ec3ab 04dbdcbf 871029c0 694a03c4
cdb1294a b9fb649c aeffdba5 ff0a4443

Private protocol key (48 octets):
5604fc7b c57eb6cc a29ef1c6 56c7e540 a16a94a6 3e367329 01829252 6b605af8
4fb3bfb5 59cadee5 7188cf02 f7578e94

Public  protocol key
x (48 octets):
d1b2e632 135a5d77 d3da21a2 71bc0e23 eab9f1c8 3c83772b f6ea988a 869d0f3b
bb066c82 4f8ab4ce 5c1d638f ac4f9b73

y (48 octets):
e4e8d9d4 d88d7e10 43d00577 c15f49a9 5bd921f8 16c946ba 89b1fbe9 46e70d1c
ebdbcc10 7520a084 b8997a01 a250a2ca

DER encoded ASN.1 (input to the SHA256 hash) (72 octets):
30463010 06072a86 48ce3d02 0106052b 81040022 03320003 b0d7d4ec bd61c96c
f246a214 5cd797dc 1d0fd3b3 539b3e8e 2ace411d 4ed736b9 5f29c73d 515cbc1c
8e9184d1 26361f9b

DER encoded ASN.1 base64 for use in the QR code (96 octets):
MEYwEAYHKoZIzj0CAQYFK4EEACIDMgADsNfU7L1hyWzyRqIUXNeX3B0P07NTmz6OKs5BHU7XNrlfKcc9UVy8HI6RhNEmNh+b

I-nonce (24 octets):
29e6194d 0a5352a7 8fac5160 2bc8c120 52f10ea9 b49f6e50
```

RESPONDER VALUES

```
Responder is: Enrollee that wants to do mutual authentication

R-capabilities (1 octets):
01

Private bootstrapping key (48 octets):
9db83151 0572f027 bfc8bf1d 6aa8222b 178a30bb 70efbe9d 55e4b9d7 421735f5
b70fee9f aa4f3410 907c37cf 546248f8

Public  bootstrapping key
x (48 octets):
6433bdc3 fd4ee707 d5026c6d f5a9b03a 2e2eb504 8c77697f ae7e2bbb ff7ed370
bf45e7bc 9d781085 87a89ed5 7d94a4b7

y (48 octets):
23fa7a95 76fd37f7 5d548a5d 115a5ecb 1b1c93ff 2bd14775 1f3282f4 266ffb8c
7e651e52 aafcc022 b25776de e84ddfcc

Private protocol key (48 octets):
452b1aac ded5852b 7d12a5ff 717449b9 4fb2911e 03b01335 2e589aa1 c6915987
144fb483 b2790335 ccc610e0 1cdadbb5

Public  protocol key
x (48 octets):
c2f24432 282c939d 05c603bb deabb1d4 cbfd7917 fd0e12a6 ab21b1aa 8111f306
```

```
431a3346 29c46973 1a19b1e9 0079d441

y (48 octets):
c38f5d0b 66598a18 51996c32 02716314 87f9405c 25e2ab1b a6eae68c c21eaeca
60b1e802 a58c5abb f27c0560 2dec587d

DER encoded ASN.1 (input to the SHA256 hash) (72 octets):
30463010 06072a86 48ce3d02 0106052b 81040022 03320002 6433bdc3 fd4ee707
d5026c6d f5a9b03a 2e2eb504 8c77697f ae7e2bbb ff7ed370 bf45e7bc 9d781085
87a89ed5 7d94a4b7

DER encoded ASN.1 base64 for use in the QR code (96 octets):
MEYwEAYHKoZIzj0CAQYFK4EEACIDMgACZDO9w/1O5wfVAmxt9amwOi4utQSMd2l/rn4ru/9+03C/Ree8nXgQhYeontV9lKS3

R-nonce (24 octets):
817bf624 f9497caf c9a195b3 45440eff aab05536 8361f6eb
```

Shared secrets

```
Secret M
x (48 octets):
f1fd1304 c0dc4461 f9c9a027 79a53650 b772da6b ab98dc47 1afb236d 41737ddb
788620b1 6a252d60 05938f50 fb3fa3d0

y (48 octets):
a8c6a12d 9dd2fb5b 958481e6 d9061a44 ead94c04 fa7ce279 a970350d 64fa959e
c9e91f3c 414e4888 bb51979c bf929026

Secret k1 (48 octets):
bd72409d 072e022e 20395d82 fc2bab99 e89d5366 99786163 c481a932 c6a9ac07
27b91916 25966ea1 3ee786a6 c5f43d23

Secret N
x (48 octets):
4d641389 d480fc87 dbac0f9b cb3eb9b6 e054ccb3 3f082613 9a7a8634 b51168db
dca4c66f 057b22bf d1e3376e ed155990

y (48 octets):
16871be5 47d7031a 58c544b5 1ab617d9 406d2b6a 331cc1c3 5a268ec0 40e5870e
ea4984b2 fb67d054 2559b3f4 8ddf923e

Secret k2 (48 octets):
c4c7eab8 ab1d8166 d097f217 fe6a3903 058ee477 cc01cdd5 36f5255a 8f28287c
60da9fc8 253a4ab7 232124bd dc2bf11c

Secret L
x (48 octets):
4884bb3d b417d046 01fdbd4d f1dd3d71 d63a9b1a 452be74b 5c3142fe 5159f296
5353361f b6fc99ac de1c3ad4 b391ffba

y (48 octets):
f3e5e99d b36b13bb cf92454e 440fc506 b6e37a1c 31db303c ae122925 3c52feb0
4fb661d4 a272779b 310dcc98 fae50219

Secret ke (48 octets):
29014e83 9ecef069 f5d96155 d900b684 63265730 7549e0e4 f48aadbd 4b302fc8
2ae2230d 9c887c3f 5c7aa060 da7606bb

I-auth (48 octets):
```

```
a73183e9 4ea589a8 c75238b7 5e390a8f 9ecd880c a15474f7 322e3217 df8d53bf
fbb9c6a1 eeffcdb1 7e5b88ac 6fa68551

R-auth (48 octets):
f9631d12 38518312 00b29539 21711dd8 b7e07c7d b6e42ad3 bacae4ab eba072c0
aa6227e2 618c7bcc 1fba71cb 08038dcd

XXXX Unknown frame duration (msec)
YYYY Unknown fragment/Sequence number
```

DPP Auth Request frame from 09:be:56:33:78:c4 to 40:c7:db:3e:74:ba

```
d000XXXX 40c7db3e 74ba09be 563378c4 40c7db3e 74baYYYY 0409506f 9a1a0100
02102000 f20bbbc3 ca7d80c4 04113eba 79516734 47ac1f32 6837e04f af5d4645
12b22f78 01102000 74f8ce71 f3a03b8a 612a599b b9380730 74bf352d 8574638f
6160c58b d7c901aa 03106000 d1b2e632 135a5d77 d3da21a2 71bc0e23 eab9f1c8
3c83772b f6ea988a 869d0f3b bb066c82 4f8ab4ce 5c1d638f ac4f9b73 e4e8d9d4
d88d7e10 43d00577 c15f49a9 5bd921f8 16c946ba 89b1fbe9 46e70d1c ebdbcc10
7520a084 b8997a01 a250a2ca 18100200 51010410 31003492 3e0194e1 6d11946e
2c813c9a 6c42e2bd c9755b9c 9dcd950b e48e5b7b 3e8034e5 11e69c90 03d1b29b
5ed5ddc0 d40e3b

Fields of the Authentication Request message (with tag and length part omitted)
Auth req  field1, Responder Bootstrap Hash (32 octets):
f20bbbc3 ca7d80c4 04113eba 79516734 47ac1f32 6837e04f af5d4645 12b22f78

Auth req  field2, Initiator Bootstrap Hash (32 octets):
74f8ce71 f3a03b8a 612a599b b9380730 74bf352d 8574638f 6160c58b d7c901aa

Auth req  field3, Initiator Protocol Key   (96 octets):
d1b2e632 135a5d77 d3da21a2 71bc0e23 eab9f1c8 3c83772b f6ea988a 869d0f3b
bb066c82 4f8ab4ce 5c1d638f ac4f9b73 e4e8d9d4 d88d7e10 43d00577 c15f49a9
5bd921f8 16c946ba 89b1fbe9 46e70d1c ebdbcc10 7520a084 b8997a01 a250a2ca

Auth req  field4, Channel                  (2 octets):
5101

Auth req  field5, Data wrapped with k1     (49 octets):
34923e01 94e16d11 946e2c81 3c9a6c42 e2bdc975 5b9c9dcd 950be48e 5b7b3e80
34e511e6 9c9003d1 b29b5ed5 ddc0d40e 3b
```

DPP Auth Response frame from 40:c7:db:3e:74:ba to 09:be:56:33:78:c4

```
d000XXXX 09be5633 78c440c7 db3e74ba 09be5633 78c4YYYY 0409506f 9a1a0101
00100100 00021020 00f20bbb c3ca7d80 c404113e ba795167 3447ac1f 326837e0
4faf5d46 4512b22f 78011020 0074f8ce 71f3a03b 8a612a59 9bb93807 3074bf35
2d857463 8f6160c5 8bd7c901 aa091060 00c2f244 32282c93 9d05c603 bbdeabb1
d4cbfd79 17fd0e12 a6ab21b1 aa8111f3 06431a33 4629c469 731a19b1 e90079d4
41c38f5d 0b66598a 1851996c 32027163 1487f940 5c25e2ab 1ba6eae6 8cc21eae
ca60b1e8 02a58c5a bbf27c05 602dec58 7d041095 00d6d98c 39283f2f 3c1c6621
c3eb1c93 c4b83ba2 19771a32 9bf0487a 0dc3dec3 fe6962c1 8f328775 fde89e69
ee4c0107 f98b96d7 4da23498 f47b2411 33714ed9 4f6e5a16 71e66e1d ce4e7051
e3b66696 152bc3c2 87263fd2 ded315d7 fca87411 b4088243 488aae8c 1f2eea55
44bf47fc 81576740 65ab51c4 a59bcb2f 3bd12f1b cab1f823 a48ce4d8 243d5f61
1241c7fe 414ea7ae 35e0
```

```
Fields of the Authentication Response message (with tag and length part omitted)
Auth resp field0, DPP_Status               (1 octets):
00

Auth resp field1, Responder Bootstrap Hash (32 octets):
f20bbbc3 ca7d80c4 04113eba 79516734 47ac1f32 6837e04f af5d4645 12b22f78

Auth resp field2, Initiator Bootstrap Hash (32 octets):
74f8ce71 f3a03b8a 612a599b b9380730 74bf352d 8574638f 6160c58b d7c901aa

Auth resp field3, Responder Protocol Key   (96 octets):
c2f24432 282c939d 05c603bb deabb1d4 cbfd7917 fd0e12a6 ab21b1aa 8111f306
431a3346 29c46973 1a19b1e9 0079d441 c38f5d0b 66598a18 51996c32 02716314
87f9405c 25e2ab1b a6eae68c c21eaeca 60b1e802 a58c5abb f27c0560 2dec587d

Auth resp field4, Data wrapped with k2     (149 octets):
d6d98c39 283f2f3c 1c6621c3 eb1c93c4 b83ba219 771a329b f0487a0d c3dec3fe
6962c18f 328775fd e89e69ee 4c0107f9 8b96d74d a23498f4 7b241133 714ed94f
6e5a1671 e66e1dce 4e7051e3 b6669615 2bc3c287 263fd2de d315d7fc a87411b4
08824348 8aae8c1f 2eea5544 bf47fc81 57674065 ab51c4a5 9bcb2f3b d12f1bca
b1f823a4 8ce4d824 3d5f6112 41c7fe41 4ea7ae35 e0
```

DPP Auth Confirm frame from 09:be:56:33:78:c4 to 40:c7:db:3e:74:ba

```
 d000XXXX 40c7db3e 74ba09be 563378c4 40c7db3e 74baYYYY 0409506f 9a1a0102
00100100 00021020 00f20bbb c3ca7d80 c404113e ba795167 3447ac1f 326837e0
4faf5d46 4512b22f 78011020 0074f8ce 71f3a03b 8a612a59 9bb93807 3074bf35
2d857463 8f6160c5 8bd7c901 aa041044 000deb27 94291b9f d58b9ad8 303764b9
d96302c5 f1a48a18 880c7f32 6b887b51 5661578c 394187be b214931a 141f8ceb
be08694d 18a4725a 83997421 a4400efe 34c81ff0 a8

Fields of the Authentication Confirm frame (with tag and length part omitted)
Auth conf field0, DPP_Status               (1 octets):
00

Auth conf field1, Responder Bootstrap Hash (32 octets):
f20bbbc3 ca7d80c4 04113eba 79516734 47ac1f32 6837e04f af5d4645 12b22f78

Auth conf field2, Initiator Bootstrap Hash (32 octets):
74f8ce71 f3a03b8a 612a599b b9380730 74bf352d 8574638f 6160c58b d7c901aa

Auth conf field3, Data wrapped with ke     (68 octets):
0deb2794 291b9fd5 8b9ad830 3764b9d9 6302c5f1 a48a1888 0c7f326b 887b5156
61578c39 4187beb2 14931a14 1f8cebbe 08694d18 a4725a83 997421a4 400efe34
c81ff0a8
```

A.4 Test vectors for DPP Authentication using P-521 for mutual authentication

This section contains test vectors for the DPP Authentication protocol using curve P-521 and mutual authentication.

```
Curve used: NIST P-521
INITIATOR VALUES
Initiator is: Configurator

I-capabilities (1 octets):
02

Private bootstrapping key (65 octets):
```

```
60c10df1 4af5ef27 f6e362d3 1bdd9eeb 44be77a3 23ba64b0 8f3f03d5 8b92cbfe
05c182a9 1660caa0 81ca3442 43c47b5a a088bcdf 738840eb 35f0218b 9f26881e
02

Public  bootstrapping key
x (66 octets):
0114896c 5614fc01 e881ab4b cecd6a46 57ca47cb 655527fe 60ed4e12 a534e442
fda79438 de7bc3a3 450f13fc 53186358 ca2e217f fa4c5624 2e340365 a0cb3481
3281

y (65 octets):
94d9773b b46f66a6 9cc7577e bb83f1c5 7a8d6ac3 0528ad39 869ab74c d23a8e53
7dad1eab 1f39be6e 5d03c6a6 94640a53 dff6f0a4 13ca312f 7d54fd56 8f08ffa6
80

Private protocol key (66 octets):
019c1c08 caaeec38 fb931894 699b095b c3ab8c1e c7ef0622 d2e3eba8 21477c8c
6fca4177 4f21166a d98aebda 37c067d9 aa08a8a2 e1b5c44c 61f2bae0 2a61f85d
9661

Public  protocol key
x (65 octets):
7cee6a53 e28ba817 e348447a 72c4ba5f 671dfacf e53c270d 31f9c9f2 8c982d02
7f2606c2 f26c868e 9b136447 8a939fd6 e116b068 aeaa98a9 eee2d153 b7f46bd6
87

y (66 octets):
01aa364e 7d6b8d32 cfd43124 dad0eb34 c5b69e3b 9298ad5f cd1fd04f 6c7fe407
8ae92703 0dde7176 7c3964c2 587e045d 2cd81197 c90ad988 318b9741 0ac021ca
d83b

DER encoded ASN.1 (input to the SHA256 hash) (90 octets):
30583010 06072a86 48ce3d02 0106052b 81040023 03440002 0114896c 5614fc01
e881ab4b cecd6a46 57ca47cb 655527fe 60ed4e12 a534e442 fda79438 de7bc3a3
450f13fc 53186358 ca2e217f fa4c5624 2e340365 a0cb3481 3281

DER encoded ASN.1 base64 for use in the QR code (120 octets):
MFgwEAYHKoZIzj0CAQYFK4EEACMDRAACARSJbFYU/AHogatLzs1qRlfKR8tlVSf+YO1OEqU05EL9p5Q43nvDo0UPE/xTGGNYyi4
hf/pMViQuNANloMs0gTKB

I-nonce (32 octets):
de972af3 847bec3b a2aedd9f 5c21cfde c7bf0bc5 fe8b276c bcd02678 07fb15b0
```

RESPONDER VALUES

```
Responder is: Enrollee that wants to do mutual authentication

R-capabilities (1 octets):
01

Private bootstrapping key (65 octets):
61e54f51 8cdf8597 35da3dd6 4c6f72c2 f086f41a 6fd52915 152ea2fe 0f24ddae
cd888373 0c9c9fd8 2cf7c043 a4102169 6388cf51 90b731dd 83638bcd 56d8b6c7
43

Public  bootstrapping key
x (66 octets):
01d38cca 93cef4cf 30c824a7 e4854d03 85ed9b1d a87d5f28 1c891e90 1a65f242
328787d3 dfcb257a f6e19577 170fbc58 2fdcda42 c6a43ff5 16e9036d 8c328ece
```

```
73de

y (65 octets):
bb5b69e9 c51dfb23 3757e927 b68046d7 7d2a1196 bc485142 1d656970 bd221bbe
52e15d3d 1cbac600 629229a0 180a87bd 9d8a834c 750a9e8e 1be9f4cc 9cb0c07d
f0

Private protocol key (66 octets):
01d8b7b1 7cd1b0a3 3f7c66fb 42209993 29cdaf4f 8b44b2ff adde8ab8 ed8abffa
9f5358c5 b1caae26 709ca4fb 78e52a4d 08f2e4f2 4111a36a 6f440d20 a0000ff5
1597

Public  protocol key
x (65 octets):
e2cacd16 10b7c147 899fbfba 55fdf336 664b80a5 a4e790ad eab1ca7b 5ca62641
f25bee4a 1974d43a b2eb4626 f8972832 80ac7cd2 e2790709 ba7d6ad9 0ce64fad
04

y (66 octets):
01d59242 92225c0c 220ac340 d1d19219 adc58fdf b9d0d911 dec8201b 513445af
b1a21001 95ae1bec e9c21f4f 28a2fe3c 0d828e93 44f6bd43 c8123fc9 d23e2c71
d223

DER encoded ASN.1 (input to the SHA256 hash) (90 octets):
30583010 06072a86 48ce3d02 0106052b 81040023 03440002 01d38cca 93cef4cf
30c824a7 e4854d03 85ed9b1d a87d5f28 1c891e90 1a65f242 328787d3 dfcb257a
f6e19577 170fbc58 2fdcda42 c6a43ff5 16e9036d 8c328ece 73de

DER encoded ASN.1 base64 for use in the QR code (120 octets):
MFgwEAYHKoZIzj0CAQYFK4EEACMDRAACAdOMypPO9M8wyCSn5IVNA4Xtmx2ofV8oHIkekBpl8kIyh4fT38slevbhlXcXD7xYL9z
aQsakP/UW6QNtjDKOznPe

R-nonce (32 octets):
d749a782 012eb0a8 595af30b 2dfc8d08 80d004eb ddb55ecc 5afbdef1 8c400e01
```

Shared secrets

```
Secret M
x (65 octets):
8f8147d6 c67169a5 95e083b1 b8fc156b eaa6d598 9144f08e 31ef6e39 ba87ae65
4203aa7a ac0905e1 a7f097f0 cdf038de 106a8032 49c83591 3fbadce9 5cfa30f5
6c

y (66 octets):
01f73a0a d8777d77 f816cc5b e2c4be7e 81d2a1a2 fcde6ce0 d05d14d5 fdc211bf
a0b82259 eb02dcb7 da694d73 6f2e7304 a6dcd95a 57f89c1c fd86cc35 7c873883
7f7c

Secret k1 (64 octets):
72a054c6 029bee30 4f280cb0 ab320d66 2fb29e64 b5ba3ee6 c3aecec4 72ef528e
b062c596 96efc361 a00be8fd 52cf24f3 1d9615be 4efea91a 33df890c 74217217

Secret N
x (66 octets):
01073b08 e838b5a0 c91de7c1 ef720ded 72ad1e6f 6f809d67 494be162 82672618
b4af8360 b9edec80 5928bb02 efc190ef b1969668 dfdf6985 13575333 a44265f4
4a22

y (65 octets):
```

```
3f4efe29 97991ef0 8f8de717 9e78e58a 803460dd 399f5746 2cbe5f93 6e31920d
2c9c8226 237b15e4 d14761e2 7456aa0a 46aef582 efe1d106 25fe4e98 5faa0ae0
93

Secret k2 (64 octets):
0e3c6f24 6cb2f5d6 5d1cac58 f853a32d 300b09b6 fe49040d f020188c 79cc9676
c3c2719e 2e75f865 733d0a01 44f09367 7f1bd0e7 24ed0c3d e2b89819 a555c3f3

Secret L
x (65 octets):
e002528b 7b9cc8ae fcefbe0d c3c74fe4 7a74d32a 9df82e7b bc22b147 0894cb19
f4d57eba 935c25c5 af6d0ac7 d0058b2e b83623ca 9076fcb7 6331eaeb de5eb557
33

y (66 octets):
010477f3 1978c7a5 6aa63053 f74b22a4 3f5ca4ed 862eb82c 12d0db0c fc188b23
6af98a48 334356cb 8d99d296 9be2f423 1ab9f64c 560b6e58 a97cefe1 777ccf1e
6803

Secret ke (64 octets):
9fd75668 dabae58b cf93eb6f b38908d5 b26ad5e3 780a76a4 cc720c7b a443f20f
89add505 8c2b7120 c34adcb4 2e732a03 a7539c4b 04ebf365 23f0ca15 c10c2a95

I-auth (64 octets):
3b9c1c05 aa79d571 a8712d29 17a72d1f c0bd4cac dd83178c bbe2dae7 f45396e9
7a249c90 91352b8a 305f5cad afa038a8 455ab968 10a92a7f 81563646 4b75de05

R-auth (64 octets):
d92bf767 551de8a4 9c7aa7fa ca702725 f379371f b37a115f 95df74fe 327cef97
1a3f0939 1f9ecc96 50b3858b 24e7f912 96f714b4 37c02abd 753f2d35 812680d4

XXXX Unknown frame duration (msec)
YYYY Unknown fragment/Sequence number
```

DPP Auth Request frame from 09:3a:73:7b:79:14 to f6:ff:00:51:4b:e5

```
d000XXXX f6ff0051 4be5093a 737b7914 f6ff0051 4be5YYYY 0409506f 9a1a0100
02102000 fd6d907d edeeef2d bee967c8 b3ef93e8 0e6e326c c62b0cf6 15599417
3481bcef 01102000 ec42eff4 f213e5ec a7a1c323 4ae9e51f 3c4d45b4 e13b1a17
e73be9a3 883adad6 03108400 007cee6a 53e28ba8 17e34844 7a72c4ba 5f671dfa
cfe53c27 0d31f9c9 f28c982d 027f2606 c2f26c86 8e9b1364 478a939f d6e116b0
68aeaa98 a9eee2d1 53b7f46b d68701aa 364e7d6b 8d32cfd4 3124dad0 eb34c5b6
9e3b9298 ad5fcd1f d04f6c7f e4078ae9 27030dde 71767c39 64c2587e 045d2cd8
1197c90a d988318b 97410ac0 21cad83b 18100200 51010410 3900c9fc e4f9b89a
02b012b6 9d076c0a a6292361 32216760 dc35e2ed 520d6b15 8d346396 b3815acb
e1e335fd 8f7741ed 31de2e4a 9f1e0d58 22759d

Fields of the Authentication Request message (with tag and length part omitted)
Auth req  field1, Responder Bootstrap Hash (32 octets):
fd6d907d edeeef2d bee967c8 b3ef93e8 0e6e326c c62b0cf6 15599417 3481bcef

Auth req  field2, Initiator Bootstrap Hash (32 octets):
ec42eff4 f213e5ec a7a1c323 4ae9e51f 3c4d45b4 e13b1a17 e73be9a3 883adad6

Auth req  field3, Initiator Protocol Key   (132 octets):
007cee6a 53e28ba8 17e34844 7a72c4ba 5f671dfa cfe53c27 0d31f9c9 f28c982d
027f2606 c2f26c86 8e9b1364 478a939f d6e116b0 68aeaa98 a9eee2d1 53b7f46b
d68701aa 364e7d6b 8d32cfd4 3124dad0 eb34c5b6 9e3b9298 ad5fcd1f d04f6c7f
e4078ae9 27030dde 71767c39 64c2587e 045d2cd8 1197c90a d988318b 97410ac0
```

```
21cad83b

Auth req  field4, Channel                  (2 octets):
5101

Auth req  field5, Data wrapped with k1     (57 octets):
c9fce4f9 b89a02b0 12b69d07 6c0aa629 23613221 6760dc35 e2ed520d 6b158d34
6396b381 5acbe1e3 35fd8f77 41ed31de 2e4a9f1e 0d582275 9d
```

DPP Auth Response frame from f6:ff:00:51:4b:e5 to 09:3a:73:7b:79:14

```
d000XXXX 093a737b 7914f6ff 00514be5 093a737b 7914YYYY 0409506f 9a1a0101
00100100 00021020 00fd6d90 7dedeeef 2dbee967 c8b3ef93 e80e6e32 6cc62b0c
f6155994 173481bc ef011020 00ec42ef f4f213e5 eca7a1c3 234ae9e5 1f3c4d45
b4e13b1a 17e73be9 a3883ada d6091084 0000e2ca cd1610b7 c147899f bfba55fd
f336664b 80a5a4e7 90adeab1 ca7b5ca6 2641f25b ee4a1974 d43ab2eb 4626f897
283280ac 7cd2e279 0709ba7d 6ad90ce6 4fad0401 d5924292 225c0c22 0ac340d1
d19219ad c58fdfb9 d0d911de c8201b51 3445afb1 a2100195 ae1bece9 c21f4f28
a2fe3c0d 828e9344 f6bd43c8 123fc9d2 3e2c71d2 230410b5 00ee42a3 7d5304f1
2450e7cc 84acfea4 95b5aabc dbef811c 6623f122 9d93f41b f9404f55 0b2b2edd
8e1be44d 272fa99b 12874551 1b66d4f6 f4ac6a62 a6258b29 7dfe78ce e136df6c
d60ca7d9 27159840 c161ce6c 695c0076 b0371344 c0c601bd 2b7b89f5 cfca1fc5
1733c573 38238a78 9d526a82 3a621c5b 8f9209bb e94246a3 ab914738 f6d84b6c
c796317b fd3e87c7 e3384028 3e4f5cfc c0bca339 b844c065 e7ecf2f9 b5d1bb1d
7aa48ece 86e92cf5 7545c31e c721

Fields of the Authentication Response message (with tag and length part omitted)
Auth resp field0, DPP_Status               (1 octets):
00

Auth resp field1, Responder Bootstrap Hash (32 octets):
fd6d907d edeeef2d bee967c8 b3ef93e8 0e6e326c c62b0cf6 15599417 3481bcef

Auth resp field2, Initiator Bootstrap Hash (32 octets):
ec42eff4 f213e5ec a7a1c323 4ae9e51f 3c4d45b4 e13b1a17 e73be9a3 883adad6

Auth resp field3, Responder Protocol Key   (132 octets):
00e2cacd 1610b7c1 47899fbf ba55fdf3 36664b80 a5a4e790 adeab1ca 7b5ca626
41f25bee 4a1974d4 3ab2eb46 26f89728 3280ac7c d2e27907 09ba7d6a d90ce64f
ad0401d5 92429222 5c0c220a c340d1d1 9219adc5 8fdfb9d0 d911dec8 201b5134
45afb1a2 100195ae 1bece9c2 1f4f28a2 fe3c0d82 8e9344f6 bd43c812 3fc9d23e
2c71d223

Auth resp field4, Data wrapped with k2     (181 octets):
ee42a37d 5304f124 50e7cc84 acfea495 b5aabcdb ef811c66 23f1229d 93f41bf9
404f550b 2b2edd8e 1be44d27 2fa99b12 8745511b 66d4f6f4 ac6a62a6 258b297d
fe78cee1 36df6cd6 0ca7d927 159840c1 61ce6c69 5c0076b0 371344c0 c601bd2b
7b89f5cf ca1fc517 33c57338 238a789d 526a823a 621c5b8f 9209bbe9 4246a3ab
914738f6 d84b6cc7 96317bfd 3e87c7e3 3840283e 4f5cfcc0 bca339b8 44c065e7
ecf2f9b5 d1bb1d7a a48ece86 e92cf575 45c31ec7 21
```

DPP Auth Confirm frame from 09:3a:73:7b:79:14 to f6:ff:00:51:4b:e5

```
d000XXXX f6ff0051 4be5093a 737b7914 f6ff0051 4be5YYYY 0409506f 9a1a0102
00100100 00021020 00fd6d90 7dedeeef 2dbee967 c8b3ef93 e80e6e32 6cc62b0c
f6155994 173481bc ef011020 00ec42ef f4f213e5 eca7a1c3 234ae9e5 1f3c4d45
b4e13b1a 17e73be9 a3883ada d6041054 00af480c 25f80558 343311fd 7550490b
56a3f162 ae980b6b 272c6a56 f3079ae5 3ce213b8 ab6aff52 16e53c18 217d42c0
```

```
ecf1637a 2a4cb4b7 0c0498f0 950a9ccf a9a77c38 a5d10da9 b2f05435 fa05de22
739b7364 77

Fields of the Authentication Confirm frame (with tag and length part omitted)
Auth conf field0, DPP_Status               (1 octets):
00

Auth conf field1, Responder Bootstrap Hash (32 octets):
fd6d907d edeeef2d bee967c8 b3ef93e8 0e6e326c c62b0cf6 15599417 3481bcef

Auth conf field2, Initiator Bootstrap Hash (32 octets):
ec42eff4 f213e5ec a7a1c323 4ae9e51f 3c4d45b4 e13b1a17 e73be9a3 883adad6

Auth conf field3, Data wrapped with ke     (84 octets):
af480c25 f8055834 3311fd75 50490b56 a3f162ae 980b6b27 2c6a56f3 079ae53c
e213b8ab 6aff5216 e53c1821 7d42c0ec f1637a2a 4cb4b70c 0498f095 0a9ccfa9
a77c38a5 d10da9b2 f05435fa 05de2273 9b736477
```

A.5 Test vectors for DPP Authentication using Brainpool P-256r1 for mutual authentication

This section contains test vectors for the DPP Authentication protocol using curve Brainpool P-256r1 and mutual authentication.

```
Curve used: Brainpool P-256r1
INITIATOR VALUES
Initiator is: Configurator

I-capabilities (1 octets):
02

Private bootstrapping key (32 octets):
8c98da1e ebfc33b2 9b6b27bb 73ebe022 2e756e1f 77652478 5221c439 5f510e9c

Public  bootstrapping key
x (32 octets):
a58dc118 c37bf1eb c9dfa675 af75a8bb 240e1200 b2765547 9e403e08 52e1cdb9

y (32 octets):
a655bf70 4d2daac8 f81493ae 39a4e3d4 3bcd2de2 1469b462 0a06dc82 d560f9fa

Private protocol key (32 octets):
0ac1c803 89c7301b 8f1131c5 90a4f945 ad5aacb4 1cb107dd e1a4fb84 5f86c93f

Public  protocol key
x (32 octets):
584c980d ad000f25 64679ac8 30a22164 462b56e1 ade91f5c 69897bab bcc4153f

y (32 octets):
062681bc 3ce67abb 3475028e c8524668 d31e708c 8ac762e4 b7cc1a8b 9bb3073c

DER encoded ASN.1 (input to the SHA256 hash) (60 octets):
303a3014 06072a86 48ce3d02 0106092b 24030302 08010107 03220002 a58dc118
c37bf1eb c9dfa675 af75a8bb 240e1200 b2765547 9e403e08 52e1cdb9

DER encoded ASN.1 base64 for use in the QR code (80 octets):
MDowFAYHKoZIzj0CAQYJKyQDAwIIAQEHAyIAAqWNwRjDe/Hryd+mda91qLskDhIAsnZVR55APghS4c25

I-nonce (16 octets):
9836bd12 59f244a5 44c2dfd6 219ec2c8
```

RESPONDER VALUES

```
Responder is: Enrollee that wants to do mutual authentication

R-capabilities (1 octets):
01

Private bootstrapping key (32 octets):
5c56e87f 1cfa3722 29b00cda 545cd3e7 44536c2d 03e7109c da895f39 11c44295

Public  bootstrapping key
x (32 octets):
a941bd0d 58d439ea 216dc5be f57ac97b 517600c0 30b186fd 4f20c596 9484920d

y (32 octets):
9af3e82e b12cdcfd 597920a1 8a865cbf 74ec6c5c d4ffd3ca 0be4af9a 1784937b

Private protocol key (32 octets):
a4da6a18 6e26ec71 5bdf3184 12841e63 2bdd52e8 82529f94 46ad58ab a1727815

Public  protocol key
x (32 octets):
a1544d0c 19108336 1db2c7d4 06abb0e1 026f116b 67099121 36cd3cf7 83f5ee7b

y (32 octets):
a02f7c9a cb3e481e 639572f2 0197d80d 8563e483 0f95bb1b ad99ed62 61c9c955

DER encoded ASN.1 (input to the SHA256 hash) (60 octets):
303a3014 06072a86 48ce3d02 0106092b 24030302 08010107 03220003 a941bd0d
58d439ea 216dc5be f57ac97b 517600c0 30b186fd 4f20c596 9484920d

DER encoded ASN.1 base64 for use in the QR code (80 octets):
MDowFAYHKoZIzj0CAQYJKyQDAwIIAQEHAyIAA6lBvQ1Y1DnqIW3FvvV6yXtRdgDAMLGG/U8gxZaUhJIN

R-nonce (16 octets):
2e34cacd bf864214 dc0c3b2b c536251b
```

Shared secrets

```
Secret M
x (32 octets):
86b40a9c 27d004ee b79d4ea7 e16c04e2 7f46219c ffe2e74f c0a57722 6d44ee04

y (32 octets):
81fe2bc3 c0d0459b c3a0f1d1 362ffea7 769a7c2c 2221d09f a98640b2 c219b03d

Secret k1 (32 octets):
afcd814c aba7ddd3 eaabc669 0d1b28ab 262c3f7b c9f64074 7c2dfe1d d2314aaf

Secret N
x (32 octets):
9103e4ef 5cdd7994 7888f47a 7227308a 83a01ef7 74b8103b 99bc1b94 6f6580ee

y (32 octets):
4eb5d3b3 0e1a6ff3 03d65b70 172864ce c1a2eb1a 9a088e56 bb9cc231 117a11c9

Secret k2 (32 octets):
```

```
ce636a02 f67efd47 fc9341f1 4f015d4d 4a60810c db67c1bb 37d8a968 081003b6

Secret L
x (32 octets):
983c085d 65d59086 50320ad4 7e631557 e0e0af4a dad4bb38 efcd39c3 f73439c5

y (32 octets):
7d23184e 789e77c6 6aa57554 c4fd3ea0 f4062117 f286ff48 01603133 552f3f1e

Secret ke (32 octets):
4a26b8fb 285c53e3 11bae79c b61ea44d 69183e75 ec0e3d4c ba83325f b8fb7885

I-auth (32 octets):
99b9054a 83f804d2 c27aa1fc 2cceeb3e 3faa6790 e5b72db4 6a10c71c bb25c6b0

R-auth (32 octets):
9809505b b3eedb8f 06c87767 58b2e1c7 343305b2 2257f284 55e86ea6 33d0f6b8

XXXX Unknown frame duration (msec)
YYYY Unknown fragment/Sequence number
```

DPP Auth Request frame from 2d:cc:e9:ce:31:1d to 3d:9e:49:a3:75:53

```
d000XXXX 3d9e49a3 75532dcc e9ce311d 3d9e49a3 7553YYYY 0409506f 9a1a0100
02102000 63cc6b7a 9312f7c7 a9c18307 10477d22 c217c590 d98855df 16cd9b14
17e08a9d 01102000 a11769e6 62550e5a 965d36de ccc86205 baf4e65e 824d813e
d59e65d9 71a21c7f 03104000 584c980d ad000f25 64679ac8 30a22164 462b56e1
ade91f5c 69897bab bcc4153f 062681bc 3ce67abb 3475028e c8524668 d31e708c
8ac762e4 b7cc1a8b 9bb3073c 18100200 51010410 29008e56 b6d1cb04 b74143b3
db7782e9 4400e6b1 cf5b9fda 4e3d128e adf80dfa 6d8f734f 37940d4e 36df64

Fields of the Authentication Request message (with tag and length part omitted)
Auth req  field1, Responder Bootstrap Hash (32 octets):
63cc6b7a 9312f7c7 a9c18307 10477d22 c217c590 d98855df 16cd9b14 17e08a9d

Auth req  field2, Initiator Bootstrap Hash (32 octets):
a11769e6 62550e5a 965d36de ccc86205 baf4e65e 824d813e d59e65d9 71a21c7f

Auth req  field3, Initiator Protocol Key   (64 octets):
584c980d ad000f25 64679ac8 30a22164 462b56e1 ade91f5c 69897bab bcc4153f
062681bc 3ce67abb 3475028e c8524668 d31e708c 8ac762e4 b7cc1a8b 9bb3073c

Auth req  field4, Channel                  (2 octets):
5101

Auth req  field5, Data wrapped with k1     (41 octets):
8e56b6d1 cb04b741 43b3db77 82e94400 e6b1cf5b 9fda4e3d 128eadf8 0dfa6d8f
734f3794 0d4e36df 64
```

DPP Auth Response frame from 3d:9e:49:a3:75:53 to 2d:cc:e9:ce:31:1d

```
d000XXXX 2dcce9ce 311d3d9e 49a37553 2dcce9ce 311dYYYY 0409506f 9a1a0101
00100100 00021020 0063cc6b 7a9312f7 c7a9c183 0710477d 22c217c5 90d98855
df16cd9b 1417e08a 9d011020 00a11769 e662550e 5a965d36 deccc862 05baf4e6
5e824d81 3ed59e65 d971a21c 7f091040 00a1544d 0c191083 361db2c7 d406abb0
e1026f11 6b670991 2136cd3c f783f5ee 7ba02f7c 9acb3e48 1e639572 f20197d8
0d8563e4 830f95bb 1bad99ed 6261c9c9 55041075 0084ece6 7fe9e373 81ef3ee3
55d5d0b2 39059868 4b069c8e c9f8a2b1 8dd74821 f959b227 4c7c2032 6bfd0a30
```

```
0d317884 e4a226df 7e68d8ef 0b250746 cf6b327e a564172f 5c59d8e1 eaffd50b
c3c75c6d 71401447 c2db7850 dd0a0afb 8ea633dd 1fe74bda 93aff92a 5de7eeae
7619e91a 812de11b 40c7

Fields of the Authentication Response message (with tag and length part omitted)
Auth resp field0, DPP_Status              (1 octets):
00

Auth resp field1, Responder Bootstrap Hash (32 octets):
63cc6b7a 9312f7c7 a9c18307 10477d22 c217c590 d98855df 16cd9b14 17e08a9d

Auth resp field2, Initiator Bootstrap Hash (32 octets):
a11769e6 62550e5a 965d36de ccc86205 baf4e65e 824d813e d59e65d9 71a21c7f

Auth resp field3, Responder Protocol Key   (64 octets):
a1544d0c 19108336 1db2c7d4 06abb0e1 026f116b 67099121 36cd3cf7 83f5ee7b
a02f7c9a cb3e481e 639572f2 0197d80d 8563e483 0f95bb1b ad99ed62 61c9c955

Auth resp field4, Data wrapped with k2     (117 octets):
84ece67f e9e37381 ef3ee355 d5d0b239 0598684b 069c8ec9 f8a2b18d d74821f9
59b2274c 7c20326b fd0a300d 317884e4 a226df7e 68d8ef0b 250746cf 6b327ea5
64172f5c 59d8e1ea ffd50bc3 c75c6d71 401447c2 db7850dd 0a0afb8e a633dd1f
e74bda93 aff92a5d e7eeae76 19e91a81 2de11b40 c7
```

DPP Auth Confirm frame from 2d:cc:e9:ce:31:1d to 3d:9e:49:a3:75:53

```
d000XXXX 3d9e49a3 75532dcc e9ce311d 3d9e49a3 7553YYYY 0409506f 9a1a0102
00100100 00021020 0063cc6b 7a9312f7 c7a9c183 0710477d 22c217c5 90d98855
df16cd9b 1417e08a 9d011020 00a11769 e662550e 5a965d36 deccc862 05baf4e6
5e824d81 3ed59e65 d971a21c 7f041034 00a73706 dd9a9faa 9098178c 5411087b
71eb21a9 fc7d4252 fd8f2a4c ff0bb58f d79fc800 7995f9ed 90cad7f6 588a2577
b88eea47 83

Fields of the Authentication Confirm frame (with tag and length part omitted)
Auth conf field0, DPP_Status              (1 octets):
00

Auth conf field1, Responder Bootstrap Hash (32 octets):
63cc6b7a 9312f7c7 a9c18307 10477d22 c217c590 d98855df 16cd9b14 17e08a9d

Auth conf field2, Initiator Bootstrap Hash (32 octets):
a11769e6 62550e5a 965d36de ccc86205 baf4e65e 824d813e d59e65d9 71a21c7f

Auth conf field3, Data wrapped with ke     (52 octets):
a73706dd 9a9faa90 98178c54 11087b71 eb21a9fc 7d4252fd 8f2a4cff 0bb58fd7
9fc80079 95f9ed90 cad7f658 8a2577b8 8eea4783
```

A.6 Test vectors for DPP Authentication using Brainpool P-384r1 using mutual authentication

This section contains test vectors for the DPP Authentication protocol using curve Brainpool P-384r1 and mutual authentication.

```
Curve used: Brainpool P-384r1
INITIATOR VALUES
Initiator is: Configurator

I-capabilities (1 octets):
02
```

```
Private bootstrapping key (48 octets):
566a411b 69d543f4 4ef5e99a 3beeb1b5 6d7d40bf 4bb14f07 63eb12c4 c9265a3a
f1af1014 09c47b8c bb95e425 79b65aa8

Public  bootstrapping key
x (48 octets):
4f210736 b9205383 a16e342f f4ea1b46 2a39f89f cf5e6715 8bcbe636 64b15d81
04122ad8 50a171bf acc53095 3aab1232

y (48 octets):
08827cb1 68d9e9a6 2b5a53d3 e6ebe60e 2b052492 92b9b939 e64f26a1 340afe1f
67377d44 44a6e811 e8cb6ac3 a39ef56e

Private protocol key (48 octets):
5a9a24d4 8cc1c804 c4e7fe33 5ca393aa 0a283eed ae0a5bd7 103ea089 152b1469
9294c7e4 403339bf e3593548 3c2d8a4e

Public  protocol key
x (48 octets):
18074104 8efc3fda 687376cd cccba188 9cd49d30 8f360b1b 9110c098 0fd5382b
b832340d 9d9f4665 05c07901 39da939e

y (48 octets):
6123577c 8ce43a2b efb130dd b7d18baa d10452a8 2ee613d0 0d548211 5bb92f8f
eafae45e 475fd7e6 3da0c03b b81af5a8

DER encoded ASN.1 (input to the SHA256 hash) (76 octets):
304a3014 06072a86 48ce3d02 0106092b 24030302 0801010b 03320002 4f210736
b9205383 a16e342f f4ea1b46 2a39f89f cf5e6715 8bcbe636 64b15d81 04122ad8
50a171bf acc53095 3aab1232

DER encoded ASN.1 base64 for use in the QR code (104 octets):
MEowFAYHKoZIzj0CAQYJKyQDAwIIAQELAzIAAk8hBza5IFODoW40L/TqG0YqOfifz15nFYvL5jZksV2BBBIq2FChcb+sxTCVOqs
SMg==

I-nonce (24 octets):
4a8ec264 9c76a056 64fa5a4a 5b0b9ae3 9aa4f991 13112b42
```

RESPONDER VALUES

```
Responder is: Enrollee that wants to do mutual authentication

R-capabilities (1 octets):
01

Private bootstrapping key (48 octets):
6a9f80cb 3a06364f bbec3cdd 3d3ee394 fedbe13f 52df0088 3cac9512 c67e9dea
933dfe86 006e1d97 2d5d646a d9071b30

Public  bootstrapping key
x (48 octets):
5d27fbd1 9dcdebcc c43f9cb1 ce9cf448 e41bddd6 5fe65b88 a20307a9 318092b8
54b4ac24 ae0d6d7f ca5ebce2 6bf47bac

y (48 octets):
1565f778 7218d3f6 ab236948 4ed3b5f7 c0795556 9825990a 29975869 99fb7bf2
6d37bf26 789294d4 920ee9ef 46698f54

Private protocol key (48 octets):
69807ec8 a016faee 99eab39b fe2c14d0 012bf651 0b83f101 1d981f97 d2f26d34
```

```
ed137f24 bf1b611b f2f7e075 4b1b3c48

Public  protocol key
x (48 octets):
6f428a34 f0c8c6f8 473fe277 6fecc43b cb939724 4d40341e fe6a7e82 a840d209
a6aa6352 e7497841 67980f78 889fe125

y (48 octets):
8ad0b71b 5ce8a82e dd82cc25 74b3aa09 458c565f 8c44b89c 3ba439d8 c674535a
a781b217 1fae45d5 67820486 0131ba9b

DER encoded ASN.1 (input to the SHA256 hash) (76 octets):
304a3014 06072a86 48ce3d02 0106092b 24030302 0801010b 03320002 5d27fbd1
9dcdebcc c43f9cb1 ce9cf448 e41bddd6 5fe65b88 a20307a9 318092b8 54b4ac24
ae0d6d7f ca5ebce2 6bf47bac

DER encoded ASN.1 base64 for use in the QR code (104 octets):
MEowFAYHKoZIzj0CAQYJKyQDAwIIAQELAzIAAl0n+9GdzevMxD+csc6c9EjkG93WX+ZbiKIDB6kxgJK4VLSsJK4NbX/KXrzia/R
7rA==

R-nonce (24 octets):
d09389c9 a4db1c61 07d34661 6561b038 971ff93a a23a4807
```

Shared secrets

```
Secret M
x (48 octets):
81c5537b b3910f91 ae2d5573 96d1dcfb 2f487ded 06f7802f 90fb3551 6623799f
577e5fde 586590aa 7f8f5c8b 7681ea4b

y (48 octets):
7f361a72 78acc5bd 27d410d8 079a3bf5 f075a994 addd2035 47022345 05c00a19
781c28cd ac467ae4 6c588dc2 969147bc


Secret k1 (48 octets):
89cae8d9 97fc0cbc 862b1988 8c32a1ef ef7920f1 b3328b36 f9343004 86cdd405
29ca639f e8e4dfd4 454e31e6 21b373b3

Secret N
x (48 octets):
54166d62 2569683f 0c58e1e3 a8360afc b80a6916 c22fac25 4230b781 fa35ddbb
8ba4a30f b8f72f39 41ac5725 7d9bb536

y (48 octets):
11220edb 1e0c907e 249bce0b 2e4d270e c1811091 adf47a9c 85154303 df1a0755
1117759c 71893ab6 0f45a55c 20c5ad4c


Secret k2 (48 octets):
b0746152 3eb24433 9c2bf1e2 c9705267 d696ca9b 4b1cbf08 9a594f8e 10a1da02
3c6d5c65 128de107 daa8a74d 472e23ec

Secret L
x (48 octets):
2a89a960 1ad05871 4b9f2420 755581ab 9162a439 32544139 180941ab f0a5a529
6199546b 06e7ce30 a5c8cdea 656da397

y (48 octets):
0816fa24 4f91a726 fd94359d ccaae1b3 1c0601b5 b93187fc 43416f83 e482ddbf
62cf1039 d7514238 a8673d5c f8d350e6
```

```
Secret ke (48 octets):
b3c43409 9b1f3a78 5cacec55 9b73abe6 b5f6e14c f5cd6486 21313936 e8aa5f9c
79af5712 d32f274e 72adb1d8 4146c73e

I-auth (48 octets):
e2f1e7d6 5626af05 2c9e3b81 0665e632 c5a1f053 18b98d93 44e5a2a9 61478e1d
947fb389 ae89b5a3 e9f7fbec 73f64e24

R-auth (48 octets):
f77008c8 78b038ec 6188cbef 7e7c8da4 fa37eccf 00f5b565 d5aeb446 8ecb2770
cc7e2993 4de202dc 8af28a95 afbe8d90


XXXX Unknown frame duration (msec)
YYYY Unknown fragment/Sequence number
```

DPP Auth Request frame from 90:90:fb:08:80:f4 to 91:53:4b:5a:18:5b

```
d000XXXX 91534b5a 185b9090 fb0880f4 91534b5a 185bYYYY 0409506f 9a1a0100
02102000 32f774e7 97dd46dc a33bbd36 26ce83a7 3e6d3241 660b4f6e d8c29f9c
cecade11 01102000 96a90fb8 90438970 55a7fc88 a9029d25 d5e45d77 b76343e1
48a4e7f5 7cce98c7 03106000 18074104 8efc3fda 687376cd cccba188 9cd49d30
8f360b1b 9110c098 0fd5382b b832340d 9d9f4665 05c07901 39da939e 6123577c
8ce43a2b efb130dd b7d18baa d10452a8 2ee613d0 0d548211 5bb92f8f eafae45e
475fd7e6 3da0c03b b81af5a8 18100200 51010410 31007247 c976e94a 8e912ceb
646d3e71 1515ecb0 969c6573 c993850f 029a1b2a 261bd9b4 f77d0d00 71562d52
2e5b3bad 78d554

Fields of the Authentication Request message (with tag and length part omitted)
Auth req  field1, Responder Bootstrap Hash (32 octets):
32f774e7 97dd46dc a33bbd36 26ce83a7 3e6d3241 660b4f6e d8c29f9c cecade11

Auth req  field2, Initiator Bootstrap Hash (32 octets):
96a90fb8 90438970 55a7fc88 a9029d25 d5e45d77 b76343e1 48a4e7f5 7cce98c7

Auth req  field3, Initiator Protocol Key   (96 octets):
18074104 8efc3fda 687376cd cccba188 9cd49d30 8f360b1b 9110c098 0fd5382b
b832340d 9d9f4665 05c07901 39da939e 6123577c 8ce43a2b efb130dd b7d18baa
d10452a8 2ee613d0 0d548211 5bb92f8f eafae45e 475fd7e6 3da0c03b b81af5a8

Auth req  field4, Channel                  (2 octets):
5101

Auth req  field5, Data wrapped with k1     (49 octets):
7247c976 e94a8e91 2ceb646d 3e711515 ecb0969c 6573c993 850f029a 1b2a261b
d9b4f77d 0d007156 2d522e5b 3bad78d5 54
```

DPP Auth Response frame from 91:53:4b:5a:18:5b to 90:90:fb:08:80:f4

```
d000XXXX 9090fb08 80f49153 4b5a185b 9090fb08 80f4YYYY 0409506f 9a1a0101
00100100 00021020 0032f774 e797dd46 dca33bbd 3626ce83 a73e6d32 41660b4f
6ed8c29f 9ccecade 11011020 0096a90f b8904389 7055a7fc 88a9029d 25d5e45d
77b76343 e148a4e7 f57cce98 c7091060 006f428a 34f0c8c6 f8473fe2 776fecc4
3bcb9397 244d4034 1efe6a7e 82a840d2 09a6aa63 52e74978 4167980f 78889fe1
258ad0b7 1b5ce8a8 2edd82cc 2574b3aa 09458c56 5f8c44b8 9c3ba439 d8c67453
5aa781b2 171fae45 d5678204 860131ba 9b041095 00d049e5 83507147 cf0e32b1
8b8ffaf4 bbc87502 9fa2230f 48901b32 0b428ba6 d0539589 94aab7a1 1fc4a8f6
c03a6497 46ac41ad 9d8e8e18 ac1b72a1 d26c1c2c 47a30718 efafc81e 8fc86ff1
dda4f0e6 b57ae6d3 d4d8446d a7539af9 7b0f16f0 7183207e b46476e2 82d4bcdc
```

```
81b2a86e 495db29d 6e15470c 8422a905 f4716cf2 851794dd d6a571bd 2d37f0dc
3fcda42f 7265ca26 1ecb

Fields of the Authentication Response message (with tag and length part omitted)
Auth resp field0, DPP_Status               (1 octets):
00

Auth resp field1, Responder Bootstrap Hash (32 octets):
32f774e7 97dd46dc a33bbd36 26ce83a7 3e6d3241 660b4f6e d8c29f9c cecade11

Auth resp field2, Initiator Bootstrap Hash (32 octets):
96a90fb8 90438970 55a7fc88 a9029d25 d5e45d77 b76343e1 48a4e7f5 7cce98c7

Auth resp field3, Responder Protocol Key   (96 octets):
6f428a34 f0c8c6f8 473fe277 6fecc43b cb939724 4d40341e fe6a7e82 a840d209
a6aa6352 e7497841 67980f78 889fe125 8ad0b71b 5ce8a82e dd82cc25 74b3aa09
458c565f 8c44b89c 3ba439d8 c674535a a781b217 1fae45d5 67820486 0131ba9b

Auth resp field4, Data wrapped with k2     (149 octets):
d049e583 507147cf 0e32b18b 8ffaf4bb c875029f a2230f48 901b320b 428ba6d0
53958994 aab7a11f c4a8f6c0 3a649746 ac41ad9d 8e8e18ac 1b72a1d2 6c1c2c47
a30718ef afc81e8f c86ff1dd a4f0e6b5 7ae6d3d4 d8446da7 539af97b 0f16f071
83207eb4 6476e282 d4bcdc81 b2a86e49 5db29d6e 15470c84 22a905f4 716cf285
1794ddd6 a571bd2d 37f0dc3f cda42f72 65ca261e cb
```

DPP Auth Confirm frame from 90:90:fb:08:80:f4 to 91:53:4b:5a:18:5b

```
d000XXXX 91534b5a 185b9090 fb0880f4 91534b5a 185bYYYY 0409506f 9a1a0102
00100100 00021020 0032f774 e797dd46 dca33bbd 3626ce83 a73e6d32 41660b4f
6ed8c29f 9ccecade 11011020 0096a90f b8904389 7055a7fc 88a9029d 25d5e45d
77b76343 e148a4e7 f57cce98 c7041044 00998434 d066887f 84855822 4ee3992d
94aaacfa a2312ff2 1ec9c8a4 7d5d457b fc43a0fd 21f3e203 70b9eabd f70f5bb9
b7ada854 994ba6ff 08a0e559 744076de fa00085c 15

Fields of the Authentication Confirm frame (with tag and length part omitted)
Auth conf field0, DPP_Status               (1 octets):
00

Auth conf field1, Responder Bootstrap Hash (32 octets):
32f774e7 97dd46dc a33bbd36 26ce83a7 3e6d3241 660b4f6e d8c29f9c cecade11

Auth conf field2, Initiator Bootstrap Hash (32 octets):
96a90fb8 90438970 55a7fc88 a9029d25 d5e45d77 b76343e1 48a4e7f5 7cce98c7

Auth conf field3, Data wrapped with ke     (68 octets):
998434d0 66887f84 8558224e e3992d94 aaacfaa2 312ff21e c9c8a47d 5d457bfc
43a0fd21 f3e20370 b9eabdf7 0f5bb9b7 ada85499 4ba6ff08 a0e55974 4076defa
00085c15
```

A.7 A.7 Test vectors for DPP Authentication using Brainpool P-512r1 for mutual authentication

This section contains test vectors for the DPP Authentication protocol using curve Brainpool P-512r1 and mutual authentication.

```
Curve used: Brainpool P-512r1
INITIATOR VALUES
Initiator is: Configurator

I-capabilities (1 octets):
02
```

```
Private bootstrapping key (64 octets):
75de2b05 0aad76f4 86102ff0 893dc921 cf2d2789 56ca854b 3613c477 e7df8edb
95208f50 4a06319c 5733beb9 c606f11a 5252d6f4 9ae01be7 a4ae87f2 9616f71c

Public  bootstrapping key
x (64 octets):
0dbc0ef6 6dc19112 dd25e55b 2231807f 7db13558 8d7d9263 0c523b0a 797a506f
c07530dc 98f1e12f 9cea3646 2aa7c309 c7ab9079 548b0ccb 6bfa00c7 e1eb3514

y (64 octets):
23574a94 4f31344e 31b70ceb 31de4554 c5d55c2a f66b8ae0 4b5ac329 5aa9c454
b99640a7 93f1046a b12059d2 4632fa1e fc7044bc 11a56b64 96930918 6df3a40d

Private protocol key (64 octets):
83ea32c1 4e739e00 d1193476 78de24c8 c4e802d8 c2180b0e 3cb30f6c 50322433
81169500 1460bad3 49deab1d c64611d3 4eaa8679 f6f94880 a558ab3c c4ede19e

Public  protocol key
x (64 octets):
6e560390 43dc3855 eb2fed6a 0504d45f b741a1dd 1fa32da3 451d90d6 b3292bd4
db7fbfb0 5babaabc 525d1dbc c43eec94 671aa764 1cc43086 11382fbe 59f239a4

y (64 octets):
0621ad8f b3ec87a4 e2531175 2f45ac76 50d18cf3 ed5b4fb7 9993758d acbdd5e2
873a1127 7bf41167 b4409df4 3d206478 946d302e 9e57ba8e 3c1bfce8 3a3e53a4

DER encoded ASN.1 (input to the SHA256 hash) (92 octets):
305a3014 06072a86 48ce3d02 0106092b 24030302 0801010d 03420003 0dbc0ef6
6dc19112 dd25e55b 2231807f 7db13558 8d7d9263 0c523b0a 797a506f c07530dc
98f1e12f 9cea3646 2aa7c309 c7ab9079 548b0ccb 6bfa00c7 e1eb3514

DER encoded ASN.1 base64 for use in the QR code (124 octets):
MFowFAYHKoZIzj0CAQYJKyQDAwIIAQENA0IAAw28DvZtwZES3SXlWyIxgH99sTVYjX2SYwxSOwp5elBvwHUw3Jjx4S+c6jZGKqf
DCcerkHlUiwzLa/oAx+HrNRQ=

I-nonce (32 octets):
8a8613e4 722fbad1 77877799 cef3cb3b 6aec9039 357039e0 ec7bb9bf 8bc2f508
```

RESPONDER VALUES

```
Responder is: Enrollee that wants to do mutual authentication

R-capabilities (1 octets):
01

Private bootstrapping key (64 octets):
a41b1891 cc0edee0 fa94706d a543ae74 004f02c6 2cb69af8 4e421e62 8e1b6447
859b63b3 3eb668ff 2ee2d111 e6f52181 9ab03d1b b7de0963 1203fe5f 36a7b161

Public  bootstrapping key
x (64 octets):
5846111d d314f1fb 04db37cb 2625b8e0 0bc63beb 28b4a050 5faedc88 5ea79879
50f5fac6 aa322458 13975d01 f30b4f74 14a9845e bd629d3c 6c21744d e540447e

y (64 octets):
91f8bd29 3e9922d8 c514978d ffba8f9e 02236dcf 59eec5f7 5fa6fb7f 26c9c3dc
c433c716 b8389c7f 9c758f8e 74126c76 36fc8a2a b126bd3f 245133ee 1cd579ca

Private protocol key (64 octets):
05b5ea50 7f73f786 2338fc6e c2edf822 6cd04868 501f97f7 b296fe0f 70584af7
```

```
f26763ce 27cd178b 7f51037a ae28f8e7 ddb94a23 899eedc6 62410858 119e86a3

Public  protocol key
x (64 octets):
59ff3517 3fe2775b ecbecfaa ca27b426 813d0624 6d9e1c84 b2fe8428 c23a8e14
9b7368f9 c48fc85b faf1ee76 56497333 18eeffbb d32c6225 864b21c0 2f96042c

y (64 octets):
06e4e4d0 d8b89dd2 cec1f188 995aeca5 b5a06431 9de5e806 af56a64d 8cc282d6
2ec29f76 cf2ecf43 ab5f0215 01a065fe 21bc9412 7cf3f7ce 102bb196 94f2195e

DER encoded ASN.1 (input to the SHA256 hash) (92 octets):
305a3014 06072a86 48ce3d02 0106092b 24030302 0801010d 03420002 5846111d
d314f1fb 04db37cb 2625b8e0 0bc63beb 28b4a050 5faedc88 5ea79879 50f5fac6
aa322458 13975d01 f30b4f74 14a9845e bd629d3c 6c21744d e540447e

DER encoded ASN.1 base64 for use in the QR code (124 octets):
MFowFAYHKoZIzj0CAQYJKyQDAwIIAQENA0IAAlhGER3TFPH7BNs3yyYluOALxjvrKLSgUF+u3Ihep5h5UPX6xqoyJFgTl10B8wt
PdBSphF69Yp08bCF0TeVARH4=

R-nonce (32 octets):
bb0f2be0 7496d7e5 3d3b9f92 ae9812fb 43b5ffa4 666e3b09 bb2100ac 73036f8f
```

Shared secrets

```
Secret M
x (64 octets):
5256fd25 019d6ccb afbc4959 ce938a50 f7029a26 e07efd28 fdd97e75 c89f1a44
ea893e5b 517e72a4 963e49fc 1e33abb9 1ec9fa94 7d868d4d e80b291f 6f62aab1

y (64 octets):
500915df 4e826a4f 045ecfdc 6c46a729 6c721859 4d6a2b01 324244e9 32aa97b8
3e2f748b 6823fed6 9a91f9f2 53f90c58 402773d5 ba3cbffb d720b3f8 5ad535db


Secret k1 (64 octets):
c63caef9 e712e36e e072d788 bce03170 9eadc766 7c545086 43569b48 c0f3589b
72541bb6 98e5bd5b 2f727ca0 68aecfbf fa66b308 25fb4180 e24c7934 34ccf570

Secret N
x (64 octets):
4b75dc91 6e6b67e2 d82a4b00 1c6b2f9a 52e06440 975cc62a 1c0c7dfa 2b8111a0
2c3b800a 0213e199 bc984bb5 b90e5c6c 3f829d74 3e729e28 9e349efa b749f7db

y (64 octets):
022e29ef 8d57fa03 b429a236 a6493958 cf2b8e7c baa61852 545bfb80 31900214
671a6351 841fda64 ff968623 dde1411f f213eda8 32eab6f9 6148d886 bb72c847


Secret k2 (64 octets):
4af21ca9 c8120811 d916d323 ed30dc5b 14910002 e8a7261d af536c4a 6cbc5b96
731104b6 4effe300 a6bfdc52 756f2fc2 bc566376 b66ca8a9 045df235 e60cca4a

Secret L
x (64 octets):
8c3bcaf5 d22cb885 e3ab3089 6a889a66 8b305689 94ad03bf 63e0ad2f e8b70320
6c7d7745 25aa1c57 cb7020a4 7fec89bb 21560b25 ff8de2ce fe208e86 2f668db3

y (64 octets):
978b0501 0c056ed5 a52e0c77 41772350 95b18d2f 0c080e67 7fd6821c fb42e827
7bafb058 71787d90 6e0b7fa9 ed5f5b04 d6dbafbb f530ffff fc40d47f 7149a14d
```

```
Secret ke (64 octets):
53e2b730 c3800783 8d4c8c4a 1461355f 57825832 ce461c25 3103fc8b f1889620
258b5fc2 80287d95 ed1de769 8b945be7 39ab411f d3ae7eb4 2810b4d1 96c0e28b

I-auth (64 octets):
2535b781 b73acde3 439abf0d 3c142b2d 296fc0bf 14e70759 af66f7c4 3cda5a76
e5d63add 2e37d009 8e8ad14c f6da4a1a 3b95d998 222a4556 d12422b1 46895216

R-auth (64 octets):
be263f28 a5b2efd5 63c2002e 834de6a4 6be6bc59 e2f4466f 3b3106ac c0ff633f
7a4e4c80 f4778d06 dbf76cee 74e4c568 bb94d9ff 88f4c8ab d02c05cf 18e55b7a


XXXX Unknown frame duration (msec)
YYYY Unknown fragment/Sequence number
```

DPP Auth Request frame from 95:f6:79:0f:a7:a2 to a5:15:97:5a:83:93

```
d000XXXX a515975a 839395f6 790fa7a2 a515975a 8393YYYY 0409506f 9a1a0100
02102000 fadf2e5f 2e0a6937 c9d2954d b578a373 2f012673 606f13d3 2c727a85
9864c2ff 01102000 0da6b62c cdb16a5b 6ef47954 792936cc e51ab5e2 1a963a14
2dab3415 aca65af7 03108000 6e560390 43dc3855 eb2fed6a 0504d45f b741a1dd
1fa32da3 451d90d6 b3292bd4 db7fbfb0 5babaabc 525d1dbc c43eec94 671aa764
1cc43086 11382fbe 59f239a4 0621ad8f b3ec87a4 e2531175 2f45ac76 50d18cf3
ed5b4fb7 9993758d acbdd5e2 873a1127 7bf41167 b4409df4 3d206478 946d302e
9e57ba8e 3c1bfce8 3a3e53a4 18100200 51010410 3900b207 36544885 ad55e5e1
fa65f59d e522a539 4e7783d3 5a2d8daa d185a837 72edff5e e8eafa80 b425224f
ce990e52 a5e308bc 8641d44d a23aff

Fields of the Authentication Request message (with tag and length part omitted)
Auth req  field1, Responder Bootstrap Hash (32 octets):
fadf2e5f 2e0a6937 c9d2954d b578a373 2f012673 606f13d3 2c727a85 9864c2ff

Auth req  field2, Initiator Bootstrap Hash (32 octets):
0da6b62c cdb16a5b 6ef47954 792936cc e51ab5e2 1a963a14 2dab3415 aca65af7

Auth req  field3, Initiator Protocol Key   (128 octets):
6e560390 43dc3855 eb2fed6a 0504d45f b741a1dd 1fa32da3 451d90d6 b3292bd4
db7fbfb0 5babaabc 525d1dbc c43eec94 671aa764 1cc43086 11382fbe 59f239a4
0621ad8f b3ec87a4 e2531175 2f45ac76 50d18cf3 ed5b4fb7 9993758d acbdd5e2
873a1127 7bf41167 b4409df4 3d206478 946d302e 9e57ba8e 3c1bfce8 3a3e53a4

Auth req  field4, Channel                 (2 octets):
5101

Auth req  field5, Data wrapped with k1     (57 octets):
b2073654 4885ad55 e5e1fa65 f59de522 a5394e77 83d35a2d 8daad185 a83772ed
ff5ee8ea fa80b425 224fce99 0e52a5e3 08bc8641 d44da23a ff
```

DPP Auth Response frame from a5:15:97:5a:83:93 to 95:f6:79:0f:a7:a2

```
d000XXXX 95f6790f a7a2a515 975a8393 95f6790f a7a2YYYY 0409506f 9a1a0101
00100100 00021020 00fadf2e 5f2e0a69 37c9d295 4db578a3 732f0126 73606f13
d32c727a 859864c2 ff011020 000da6b6 2ccdb16a 5b6ef479 54792936 cce51ab5
e21a963a 142dab34 15aca65a f7091080 0059ff35 173fe277 5becbecf aaca27b4
26813d06 246d9e1c 84b2fe84 28c23a8e 149b7368 f9c48fc8 5bfaf1ee 76564973
3318eeff bbd32c62 25864b21 c02f9604 2c06e4e4 d0d8b89d d2cec1f1 88995aec
a5b5a064 319de5e8 06af56a6 4d8cc282 d62ec29f 76cf2ecf 43ab5f02 1501a065
fe21bc94 127cf3f7 ce102bb1 9694f219 5e0410b5 00558356 9cd212ca 0d7bbe1a
```

```
2e597717 1ec90791 5d4bdfe3 38c0de67 f0f77341 eeb07434 9bc363b3 6141b06c
8cab43a0 197bfe4f 08c288a5 444c96de f1df5a29 20b74612 b4e5361e 121603c3
e5a909c5 be600897 7890c30a 40fea47e d8f57a65 56833eeb 08337abc 6a557daf
88ee0770 65ed2159 fa0ee88b 024871fd 935c81a3 ab819ea7 d452a449 b5eff576
cbb96297 4cefcec5 9dad3e85 337db150 c5755ed9 a4f72c2c 61d57bdf c70e99fa
2721f9dc c03ea9d1 7795

Fields of the Authentication Response message (with tag and length part omitted)
Auth resp field0, DPP_Status              (1 octets):
00

Auth resp field1, Responder Bootstrap Hash (32 octets):
fadf2e5f 2e0a6937 c9d2954d b578a373 2f012673 606f13d3 2c727a85 9864c2ff

Auth resp field2, Initiator Bootstrap Hash (32 octets):
0da6b62c cdb16a5b 6ef47954 792936cc e51ab5e2 1a963a14 2dab3415 aca65af7

Auth resp field3, Responder Protocol Key   (128 octets):
59ff3517 3fe2775b ecbecfaa ca27b426 813d0624 6d9e1c84 b2fe8428 c23a8e14
9b7368f9 c48fc85b faf1ee76 56497333 18eeffbb d32c6225 864b21c0 2f96042c
06e4e4d0 d8b89dd2 cec1f188 995aeca5 b5a06431 9de5e806 af56a64d 8cc282d6
2ec29f76 cf2ecf43 ab5f0215 01a065fe 21bc9412 7cf3f7ce 102bb196 94f2195e

Auth resp field4, Data wrapped with k2     (181 octets):
5583569c d212ca0d 7bbe1a2e 5977171e c907915d 4bdfe338 c0de67f0 f77341ee
b074349b c363b361 41b06c8c ab43a019 7bfe4f08 c288a544 4c96def1 df5a2920
b74612b4 e5361e12 1603c3e5 a909c5be 60089778 90c30a40 fea47ed8 f57a6556
833eeb08 337abc6a 557daf88 ee077065 ed2159fa 0ee88b02 4871fd93 5c81a3ab
819ea7d4 52a449b5 eff576cb b962974c efcec59d ad3e8533 7db150c5 755ed9a4
f72c2c61 d57bdfc7 0e99fa27 21f9dcc0 3ea9d177 95
```

DPP Auth Confirm frame from 95:f6:79:0f:a7:a2 to a5:15:97:5a:83:93

```
d000XXXX a515975a 839395f6 790fa7a2 a515975a 8393YYYY 0409506f 9a1a0102
00100100 00021020 00fadf2e 5f2e0a69 37c9d295 4db578a3 732f0126 73606f13
d32c727a 859864c2 ff011020 000da6b6 2ccdb16a 5b6ef479 54792936 cce51ab5
e21a963a 142dab34 15aca65a f7041054 00fcfa09 ec6bc0dc 4f129533 e795bae0
32ec7f3b 48f5fcc2 f1abd42c 8be48a11 94b4a4bc adad25b9 2bf88b37 669f5706
7565679d 2cd4e406 e38aaecd 958a593f 27034838 3f28e131 b93c3a5e b85cec81
3d72b020 42

Fields of the Authentication Confirm frame (with tag and length part omitted)
Auth conf field0, DPP_Status              (1 octets):
00

Auth conf field1, Responder Bootstrap Hash (32 octets):
fadf2e5f 2e0a6937 c9d2954d b578a373 2f012673 606f13d3 2c727a85 9864c2ff

Auth conf field2, Initiator Bootstrap Hash (32 octets):
0da6b62c cdb16a5b 6ef47954 792936cc e51ab5e2 1a963a14 2dab3415 aca65af7

Auth conf field3, Data wrapped with ke     (84 octets):
fcfa09ec 6bc0dc4f 129533e7 95bae032 ec7f3b48 f5fcc2f1 abd42c8b e48a1194
b4a4bcad ad25b92b f88b3766 9f570675 65679d2c d4e406e3 8aaecd95 8a593f27
0348383f 28e131b9 3c3a5eb8 5cec813d 72b02042
```

Appendix B Role-specific elements for PKEX

B.1 Role-specific elements for NIST p256

```
unsigned char nist_p256_initiator_x_coord[32] = {
       0x56, 0x26, 0x12, 0xcf, 0x36, 0x48, 0xfe, 0x0b,
       0x07, 0x04, 0xbb, 0x12, 0x22, 0x50, 0xb2, 0x54,
       0xb1, 0x94, 0x64, 0x7e, 0x54, 0xce, 0x08, 0x07,
       0x2e, 0xec, 0xca, 0x74, 0x5b, 0x61, 0x2d, 0x25
       };

unsigned char nist_p256_initiator_y_coord[32] = {
       0x3e, 0x44, 0xc7, 0xc9, 0x8c, 0x1c, 0xa1, 0x0b,
       0x20, 0x09, 0x93, 0xb2, 0xfd, 0xe5, 0x69, 0xdc,
       0x75, 0xbc, 0xad, 0x33, 0xc1, 0xe7, 0xc6, 0x45,
       0x4d, 0x10, 0x1e, 0x6a, 0x3d, 0x84, 0x3c, 0xa4
       };

unsigned char nist_p256_responder_x_coord[32] = {
       0x1e, 0xa4, 0x8a, 0xb1, 0xa4, 0xe8, 0x42, 0x39,
       0xad, 0x73, 0x07, 0xf2, 0x34, 0xdf, 0x57, 0x4f,
       0xc0, 0x9d, 0x54, 0xbe, 0x36, 0x1b, 0x31, 0x0f,
       0x59, 0x91, 0x52, 0x33, 0xac, 0x19, 0x9d, 0x76
       };

unsigned char nist_p256_responder_y_coord[32] = {
       0xd9, 0xfb, 0xf6, 0xb9, 0xf5, 0xfa, 0xdf, 0x19,
       0x58, 0xd8, 0x3e, 0xc9, 0x89, 0x7a, 0x35, 0xc1,
       0xbd, 0xe9, 0x0b, 0x77, 0x7a, 0xcb, 0x91, 0x2a,
       0xe8, 0x21, 0x3f, 0x47, 0x52, 0x02, 0x4d, 0x67
       };
```

B.2 Role-specific elements for NIST p384

```
unsigned char nist_p384_initiator_x_coord[48] = {
       0x95, 0x3f, 0x42, 0x9e, 0x50, 0x7f, 0xf9, 0xaa,
       0xac, 0x1a, 0xf2, 0x85, 0x2e, 0x64, 0x91, 0x68,
       0x64, 0xc4, 0x3c, 0xb7, 0x5c, 0xf8, 0xc9, 0x53,
       0x6e, 0x58, 0x4c, 0x7f, 0xc4, 0x64, 0x61, 0xac,
       0x51, 0x8a, 0x6f, 0xfe, 0xab, 0x74, 0xe6, 0x12,
       0x81, 0xac, 0x38, 0x5d, 0x41, 0xe6, 0xb9, 0xa3
       };

unsigned char nist_p384_initiator_y_coord[48] = {
       0x76, 0x2f, 0x68, 0x84, 0xa6, 0xb0, 0x59, 0x29,
       0x83, 0xa2, 0x6c, 0xa4, 0x6c, 0x3b, 0xf8, 0x56,
       0x76, 0x11, 0x2a, 0x32, 0x90, 0xbd, 0x07, 0xc7,
       0x37, 0x39, 0x9d, 0xdb, 0x96, 0xf3, 0x2b, 0xb6,
       0x27, 0xbb, 0x29, 0x3c, 0x17, 0x33, 0x9d, 0x94,
       0xc3, 0xda, 0xac, 0x46, 0xb0, 0x8e, 0x07, 0x18
       };

unsigned char nist_p384_responder_x_coord[48] = {
       0xad, 0xbe, 0xd7, 0x1d, 0x3a, 0x71, 0x64, 0x98,
       0x5f, 0xb4, 0xd6, 0x4b, 0x50, 0xd0, 0x84, 0x97,
       0x4b, 0x7e, 0x57, 0x70, 0xd2, 0xd9, 0xf4, 0x92,
       0x2a, 0x3f, 0xce, 0x99, 0xc5, 0x77, 0x33, 0x44,
       0x14, 0x56, 0x92, 0xcb, 0xae, 0x46, 0x64, 0xdf,
       0xe0, 0xbb, 0xd7, 0xb1, 0x29, 0x20, 0x72, 0xdf
```

```
};

unsigned char nist_p384_responder_y_coord[48] = {
       0xab, 0xa7, 0xdf, 0x52, 0xaa, 0xe2, 0x35, 0x0c,
       0xe3, 0x75, 0x32, 0xe6, 0xbf, 0x06, 0xc8, 0x7c,
       0x38, 0x29, 0x4c, 0xec, 0x82, 0xac, 0xd7, 0xa3,
       0x09, 0xd2, 0x0e, 0x22, 0x5a, 0x74, 0x52, 0xa1,
       0x7e, 0x54, 0x4e, 0xfe, 0xc6, 0x29, 0x33, 0x63,
       0x15, 0xe1, 0x7b, 0xe3, 0x40, 0x1c, 0xca, 0x06
       };
```

B.3 Role-specific elements for NIST p521

```
unsigned char nist_p521_initiator_x_coord[66] = {
       0x00, 0x16, 0x20, 0x45, 0x19, 0x50, 0x95, 0x23,
       0x0d, 0x24, 0xbe, 0x00, 0x87, 0xdc, 0xfa, 0xf0,
       0x58, 0x9a, 0x01, 0x60, 0x07, 0x7a, 0xca, 0x76,
       0x01, 0xab, 0x2d, 0x5a, 0x46, 0xcd, 0x2c, 0xb5,
       0x11, 0x9a, 0xff, 0xaa, 0x48, 0x04, 0x91, 0x38,
       0xcf, 0x86, 0xfc, 0xa4, 0xa5, 0x0f, 0x47, 0x01,
       0x80, 0x1b, 0x30, 0xa3, 0xae, 0xe8, 0x1c, 0x2e,
       0xea, 0xcc, 0xf0, 0x03, 0x9f, 0x77, 0x4c, 0x8d,
       0x97, 0x76
       };

unsigned char nist_p521_initiator_y_coord[66] = {
       0x00, 0xb3, 0x8e, 0x02, 0xe4, 0x2a, 0x63, 0x59,
       0x12, 0xc6, 0x10, 0xba, 0x3a, 0xf9, 0x02, 0x99,
       0x3f, 0x14, 0xf0, 0x40, 0xde, 0x5c, 0xc9, 0x8b,
       0x02, 0x55, 0xfa, 0x91, 0xb1, 0xcc, 0x6a, 0xbd,
       0xe5, 0x62, 0xc0, 0xc5, 0xe3, 0xa1, 0x57, 0x9f,
       0x08, 0x1a, 0xa6, 0xe2, 0xf8, 0x55, 0x90, 0xbf,
       0xf5, 0xa6, 0xc3, 0xd8, 0x52, 0x1f, 0xb7, 0x02,
       0x2e, 0x7c, 0xc8, 0xb3, 0x20, 0x1e, 0x79, 0x8d,
       0x03, 0xa8
       };

unsigned char nist_p521_responder_x_coord[66] = {
       0x00, 0x79, 0xe4, 0x4d, 0x6b, 0x5e, 0x12, 0x0a,
       0x18, 0x2c, 0xb3, 0x05, 0x77, 0x0f, 0xc3, 0x44,
       0x1a, 0xcd, 0x78, 0x46, 0x14, 0xee, 0x46, 0x3f,
       0xab, 0xc9, 0x59, 0x7c, 0x85, 0xa0, 0xc2, 0xfb,
       0x02, 0x32, 0x99, 0xde, 0x5d, 0xe1, 0x0d, 0x48,
       0x2d, 0x71, 0x7d, 0x8d, 0x3f, 0x61, 0x67, 0x9e,
       0x2b, 0x8b, 0x12, 0xde, 0x10, 0x21, 0x55, 0x0a,
       0x5b, 0x2d, 0xe8, 0x05, 0x09, 0xf6, 0x20, 0x97,
       0x84, 0xb4
       };

unsigned char nist_p521_responder_y_coord[66] = {
       0x00, 0x46, 0x63, 0x39, 0xbe, 0xcd, 0xa4, 0x2d,
       0xca, 0x27, 0x74, 0xd4, 0x1b, 0x91, 0x33, 0x20,
       0x83, 0xc7, 0x3b, 0xa4, 0x09, 0x8b, 0x8e, 0xa3,
       0x88, 0xe9, 0x75, 0x7f, 0x56, 0x7b, 0x38, 0x84,
       0x62, 0x02, 0x7c, 0x90, 0x51, 0x07, 0xdb, 0xe9,
       0xd0, 0xde, 0xda, 0x9a, 0x5d, 0xe5, 0x94, 0xd2,
       0xcf, 0x9d, 0x4c, 0x33, 0x91, 0xa6, 0xc3, 0x80,
       0xa7, 0x6e, 0x7e, 0x8d, 0xf8, 0x73, 0x6e, 0x53,
       0xce, 0xe1
       };
```

B.4 Role-specific elements for Brainpool p256r1

```
unsigned char brainpool_p256_initiator_x_coord[32] = {
       0x46, 0x98, 0x18, 0x6c, 0x27, 0xcd, 0x4b, 0x10,
       0x7d, 0x55, 0xa3, 0xdd, 0x89, 0x1f, 0x9f, 0xca,
       0xc7, 0x42, 0x5b, 0x8a, 0x23, 0xed, 0xf8, 0x75,
       0xac, 0xc7, 0xe9, 0x8d, 0xc2, 0x6f, 0xec, 0xd8
       };

unsigned char brainpool_p256_initiator_y_coord[32] = {
       0x93, 0xca, 0xef, 0xa9, 0x66, 0x3e, 0x87, 0xcd,
       0x52, 0x6e, 0x54, 0x13, 0xef, 0x31, 0x67, 0x30,
       0x15, 0x13, 0x9d, 0x6d, 0xc0, 0x95, 0x32, 0xbe,
       0x4f, 0xab, 0x5d, 0xf7, 0xbf, 0x5e, 0xaa, 0x0b
       };

unsigned char brainpool_p256_responder_x_coord[32] = {
       0x90, 0x18, 0x84, 0xc9, 0xdc, 0xcc, 0xb5, 0x2f,
       0x4a, 0x3f, 0x4f, 0x18, 0x0a, 0x22, 0x56, 0x6a,
       0xa9, 0xef, 0xd4, 0xe6, 0xc3, 0x53, 0xc2, 0x1a,
       0x23, 0x54, 0xdd, 0x08, 0x7e, 0x10, 0xd8, 0xe3
       };

unsigned char brainpool_p256_responder_y_coord[32] = {
       0x2a, 0xfa, 0x98, 0x9b, 0xe3, 0xda, 0x30, 0xfd,
       0x32, 0x28, 0xcb, 0x66, 0xfb, 0x40, 0x7f, 0xf2,
       0xb2, 0x25, 0x80, 0x82, 0x44, 0x85, 0x13, 0x7e,
       0x4b, 0xb5, 0x06, 0xc0, 0x03, 0x69, 0x23, 0x64
       };
```

B.5 Role-specific elements for Brainpool p384r1

```
unsigned char brainpool_p384_initiator_x_coord[48] = {
       0x0a, 0x2c, 0xeb, 0x49, 0x5e, 0xb7, 0x23, 0xbd,
       0x20, 0x5b, 0xe0, 0x49, 0xdf, 0xcf, 0xcf, 0x19,
       0x37, 0x36, 0xe1, 0x2f, 0x59, 0xdb, 0x07, 0x06,
       0xb5, 0xeb, 0x2d, 0xae, 0xc2, 0xb2, 0x38, 0x62,
       0xa6, 0x73, 0x09, 0xa0, 0x6c, 0x0a, 0xa2, 0x30,
       0x99, 0xeb, 0xf7, 0x1e, 0x47, 0xb9, 0x5e, 0xbe
       };
unsigned char brainpool_p384_initiator_y_coord[48] = {
       0x54, 0x76, 0x61, 0x65, 0x75, 0x5a, 0x2f, 0x99,
       0x39, 0x73, 0xca, 0x6c, 0xf9, 0xf7, 0x12, 0x86,
       0x54, 0xd5, 0xd4, 0xad, 0x45, 0x7b, 0xbf, 0x32,
       0xee, 0x62, 0x8b, 0x9f, 0x52, 0xe8, 0xa0, 0xc9,
       0xb7, 0x9d, 0xd1, 0x09, 0xb4, 0x79, 0x1c, 0x3e,
       0x1a, 0xbf, 0x21, 0x45, 0x66, 0x6b, 0x02, 0x52
       };

unsigned char brainpool_p384_responder_x_coord[48] = {
       0x03, 0xa2, 0x57, 0xef, 0xe8, 0x51, 0x21, 0xa0,
       0xc8, 0x9e, 0x21, 0x02, 0xb5, 0x9a, 0x36, 0x25,
       0x74, 0x22, 0xd1, 0xf2, 0x1b, 0xa8, 0x9a, 0x9b,
       0x97, 0xbc, 0x5a, 0xeb, 0x26, 0x15, 0x09, 0x71,
       0x77, 0x59, 0xec, 0x8b, 0xb7, 0xe1, 0xe8, 0xce,
       0x65, 0xb8, 0xaf, 0xf8, 0x80, 0xae, 0x74, 0x6c
       };

unsigned char brainpool_p384_responder_y_coord[48] = {
       0x2f, 0xd9, 0x6a, 0xc7, 0x3e, 0xec, 0x76, 0x65,
```

```
       0x2d, 0x38, 0x7f, 0xec, 0x63, 0x26, 0x3f, 0x04,
       0xd8, 0x4e, 0xff, 0xe1, 0x0a, 0x51, 0x74, 0x70,
       0xe5, 0x46, 0x63, 0x7f, 0x5c, 0xc0, 0xd1, 0x7c,
       0xfb, 0x2f, 0xea, 0xe2, 0xd8, 0x0f, 0x84, 0xcb,
       0xe9, 0x39, 0x5c, 0x64, 0xfe, 0xcb, 0x2f, 0xf1
       };
```

B.6 Role-specific elements for Brainpool p512r1

```
unsigned char brainpool_p512_initiator_x_coord[64] = {
       0x4c, 0xe9, 0xb6, 0x1c, 0xe2, 0x00, 0x3c, 0x9c,
       0xa9, 0xc8, 0x56, 0x52, 0xaf, 0x87, 0x3e, 0x51,
       0x9c, 0xbb, 0x15, 0x31, 0x1e, 0xc1, 0x05, 0xfc,
       0x7c, 0x77, 0xd7, 0x37, 0x61, 0x27, 0xd0, 0x95,
       0x98, 0xee, 0x5d, 0xa4, 0x3d, 0x09, 0xdb, 0x3d,
       0xfa, 0x89, 0x9e, 0x7f, 0xa6, 0xa6, 0x9c, 0xff,
       0x83, 0x5c, 0x21, 0x6c, 0x3e, 0xf2, 0xfe, 0xdc,
       0x63, 0xe4, 0xd1, 0x0e, 0x75, 0x45, 0x69, 0x0f
       };

unsigned char brainpool_p512_initiator_y_coord[64] = {
       0x50, 0xb5, 0x9b, 0xfa, 0x45, 0x67, 0x75, 0x94,
       0x44, 0xe7, 0x68, 0xb0, 0xeb, 0x3e, 0xb3, 0xb8,
       0xf9, 0x99, 0x05, 0xef, 0xae, 0x6c, 0xbc, 0xe3,
       0xe1, 0xd2, 0x51, 0x54, 0xdf, 0x59, 0xd4, 0x45,
       0x41, 0x3a, 0xa8, 0x0b, 0x76, 0x32, 0x44, 0x0e,
       0x07, 0x60, 0x3a, 0x6e, 0xbe, 0xfe, 0xe0, 0x58,
       0x52, 0xa0, 0xaa, 0x8b, 0xd8, 0x5b, 0xf2, 0x71,
       0x11, 0x9a, 0x9e, 0x8f, 0x1a, 0xd1, 0xc9, 0x99
       };

unsigned char brainpool_p512_responder_x_coord[64] = {
       0x2a, 0x60, 0x32, 0x27, 0xa1, 0xe6, 0x94, 0x72,
       0x1c, 0x48, 0xbe, 0xc5, 0x77, 0x14, 0x30, 0x76,
       0xe4, 0xbf, 0xf7, 0x7b, 0xc5, 0xfd, 0xdf, 0x19,
       0x1e, 0x0f, 0xdf, 0x1c, 0x40, 0xfa, 0x34, 0x9e,
       0x1f, 0x42, 0x24, 0xa3, 0x2c, 0xd5, 0xc7, 0xc9,
       0x7b, 0x47, 0x78, 0x96, 0xf1, 0x37, 0x0e, 0x88,
       0xcb, 0xa6, 0x52, 0x29, 0xd7, 0xa8, 0x38, 0x29,
       0x8e, 0x6e, 0x23, 0x47, 0xd4, 0x4b, 0x70, 0x3e
       };

unsigned char brainpool_p512_responder_y_coord[64] = {
       0x80, 0x1f, 0x43, 0xd2, 0x17, 0x35, 0xec, 0x81,
       0xd9, 0x4b, 0xdc, 0x81, 0x19, 0xd9, 0x5f, 0x68,
       0x16, 0x84, 0xfe, 0x63, 0x4b, 0x8d, 0x5d, 0xaa,
       0x88, 0x4a, 0x47, 0x48, 0xd4, 0xea, 0xab, 0x7d,
       0x6a, 0xbf, 0xe1, 0x28, 0x99, 0x6a, 0x87, 0x1c,
       0x30, 0xb4, 0x44, 0x2d, 0x75, 0xac, 0x35, 0x09,
       0x73, 0x24, 0x3d, 0xb4, 0x43, 0xb1, 0xc1, 0x56,
       0x56, 0xad, 0x30, 0x87, 0xf4, 0xc3, 0x00, 0xc7
       };
```

Appendix C PKEX Test Vector for NIST p256

C.1 Initial state of Initiator and Responder

```
Initiator MAC address: ac:64:91:f4:52:07
Responder MAC address: 6e:5e:ce:6e:f3:dd

code identifier: "joes_key"
password: "thisisreallysecret"

Initiator bootstrapping keys

bootstrapping private key:
5941b51a cfc702cd c1c34726 4beb2920 db88eb1a 0bf03a21 1868b163 2233c269

bootstrapping public key
A.x:
0ad58864 754c8126 85ff3a52 a573c1d7 2c72c4eb ed98f391 5622d4df c84a438d

A.y:
7e81429a ac49ddec 75ad6521 db9c7407 4e30b5eb 2ba53693 c9341b79 be14e101

Responder bootstrapping keys

bootstrapping private key:
2ae89562 93f49986 b6d0b816 9a86805d 9232babb 5f6813fd fe96f19d 59536c60

bootstrapping public key
B.x:
977b7fa3 9779a814 29febb12 e1dc5e20 a7e017c4 bc743709 0e57c966 a2b0e8a3

B.y:
9d2b6273 39476397 63f64c7b 6708c1e0 857becb7 e24fc195 248b5b06 036cf792
```

C.2 Initiator generates PKEX Exchange Request frame

```
identifier:
6a6f6573 5f6b6579

password:
74686973 69737265 616c6c79 73656372 6574

H(mymac | [identifier | ] password):
fd24178b f56fbb4a a17d18a5 5a4e7ce6 9d62cb63 2462c236 0751f507 25bc252c

Pinit.x:
562612cf 3648fe0b 0704bb12 2250b254 b194647e 54ce0807 2eecca74 5b612d25

Qi.x:
2867c4e0 80980dba d5099a8f 821e8729 679c5c71 4888c0bd 9c7e8e40 48c5fa5e

X.x:
740ab9f0 c173507b 0081b475 b275de6a 3060cf43 4b6a65f0 b0144a1d bf913310

M.x:
bcca8e23 e5c05032 ae6051ca 6392f7c4 a4b4f9fe 13e81261 32d070e5 52848176

M.y:
```

```
8fb6c57d 856ff3fe 63123a1f f80696c4 9718b4a9 c1cc11a6 29a5a73a e3ad9d78

DPP PKEX Exchange Request frame from ac:64:91:f4:52:07 to ff:ff:ff:ff:ff:ff

71876f5a 57830000 7a000000 7a000000 14000000 02000000 d0000000 ffffffff
ffffac64 91f45207 ffffffff ffff0000 0409506f 9a1a0107 12100200 13001510
08006a6f 65735f6b 65791310 4000bcca 8e23e5c0 5032ae60 51ca6392 f7c4a4b4
f9fe13e8 126132d0 70e55284 81768fb6 c57d856f f3fe6312 3a1ff806 96c49718
b4a9c1cc 11a629a5 a73ae3ad 9d78

Finite Cyclic Group, length 2, value:
1300

PKEX Code Identifier, length 8, value:
6a6f6573 5f6b6579

Encrypted Key, length 64, value:
bcca8e23 e5c05032 ae6051ca 6392f7c4 a4b4f9fe 13e81261 32d070e5 52848176
8fb6c57d 856ff3fe 63123a1f f80696c4 9718b4a9 c1cc11a6 29a5a73a e3ad9d78
```

C.3 Responder processes PKEX Exchange Request frame

```
M.x:
bcca8e23 e5c05032 ae6051ca 6392f7c4 a4b4f9fe 13e81261 32d070e5 52848176

M.y:
8fb6c57d 856ff3fe 63123a1f f80696c4 9718b4a9 c1cc11a6 29a5a73a e3ad9d78

identifier:
6a6f6573 5f6b6579

H(peermac | [identifier | ] password):
fd24178b f56fbb4a a17d18a5 5a4e7ce6 9d62cb63 2462c236 0751f507 25bc252c

Pinit.x:
562612cf 3648fe0b 0704bb12 2250b254 b194647e 54ce0807 2eecca74 5b612d25

X.x:
740ab9f0 c173507b 0081b475 b275de6a 3060cf43 4b6a65f0 b0144a1d bf913310
```

C.4 Responder generates PKEX Exchange Response frame

```
identifier:
6a6f6573 5f6b6579

password:
74686973 69737265 616c6c79 73656372 6574

H(mymac | [identifier | ] password):
ba15285a 4b2055f4 0d1a0a40 64461006 d4852ec6 c4331c19 a1562ba8 79cb2753

Presp.x:
1ea48ab1 a4e84239 ad7307f2 34df574f c09d54be 361b310f 59915233 ac199d76

Qr.x:
134af1c4 1c8e7d97 4c647cc2 bfca30b0 36966959 f9044e90 f673d756 706e624c

Y.x:
a9972a94 f143740d f31c7a61 124d01a4 e949d0fd cede6136 9f4c6b09 7aeb18b5
```

```
N.x:
0a91e072 8809bb81 91ea36d0 a1d5602b f36ab670 8fbfd063 e2511e53 3b534020

N.y:
79c4cbe7 cdea091a b934f011 4162468e ad01790a 258cbf06 62b31fd3 1ee302d9

DPP PKEX Exchange Response frame from 6e:5e:ce:6e:f3:dd to ac:64:91:f4:52:07

71876f5a 61be0000 79000000 79000000 14000000 02000000 d0000000 ac6491f4
52076e5e ce6ef3dd ffffffff ffff0000 0409506f 9a1a0108 00100100 00151008
006a6f65 735f6b65 79131040 000a91e0 728809bb 8191ea36 d0a1d560 2bf36ab6
708fbfd0 63e2511e 533b5340 2079c4cb e7cdea09 1ab934f0 11416246 8ead0179
0a258cbf 0662b31f d31ee302 d9

Status, length 1, value:
00

PKEX Code Identifier, length 8, value:
6a6f6573 5f6b6579

Encrypted Key, length 64, value:
0a91e072 8809bb81 91ea36d0 a1d5602b f36ab670 8fbfd063 e2511e53 3b534020
79c4cbe7 cdea091a b934f011 4162468e ad01790a 258cbf06 62b31fd3 1ee302d9
```

C.5 Initiator processess PKEX Exchange Response frame

```
N.x:
0a91e072 8809bb81 91ea36d0 a1d5602b f36ab670 8fbfd063 e2511e53 3b534020

N.y:
79c4cbe7 cdea091a b934f011 4162468e ad01790a 258cbf06 62b31fd3 1ee302d9

identifier:
6a6f6573 5f6b6579

H(peermac | [identifier | ] password):
ba15285a 4b2055f4 0d1a0a40 64461006 d4852ec6 c4331c19 a1562ba8 79cb2753

Presp.x:
1ea48ab1 a4e84239 ad7307f2 34df574f c09d54be 361b310f 59915233 ac199d76

Y.x:
a9972a94 f143740d f31c7a61 124d01a4 e949d0fd cede6136 9f4c6b09 7aeb18b5
```

C.6 Initiator generates PKEX Commit/Reveal request

```
x:
8365c5ed 93d751be f2d92b41 0dc6adfd 95670889 183fac1b d66759ad 85c3187a

Y.x:
a9972a94 f143740d f31c7a61 124d01a4 e949d0fd cede6136 9f4c6b09 7aeb18b5

(x*Y).x:
7415e1c6 8611f044 3cc345d1 36984e48 8c6a26d3 d5482fa6 7e9841a0 3a87c78f

context to create z
MAC-Initiator:
ac6491f4 5207

MAC-Responder:
```

```
6e5ece6e f3dd

M.x:
bcca8e23 e5c05032 ae6051ca 6392f7c4 a4b4f9fe 13e81261 32d070e5 52848176

N.x:
0a91e072 8809bb81 91ea36d0 a1d5602b f36ab670 8fbfd063 e2511e53 3b534020

password:
74686973 69737265 616c6c79 73656372 6574

z:
5271dee9 15cf7b19 08747d8e db839444 2411c518 3ee38b79 ebef399c 08738e0b

(a*Y).x:
31c1b9ab 31d9c2f2 78b35b5c 29d180df eaf76d58 5ede9c0d d91cb661 49db572e

context for u
MAC-Initiator:
ac6491f4 5207

A.x:
0ad58864 754c8126 85ff3a52 a573c1d7 2c72c4eb ed98f391 5622d4df c84a438d

Y.x:
a9972a94 f143740d f31c7a61 124d01a4 e949d0fd cede6136 9f4c6b09 7aeb18b5

X.x:
740ab9f0 c173507b 0081b475 b275de6a 3060cf43 4b6a65f0 b0144a1d bf913310

u:
598c3d8d cccea2d4 3259068d 542a9074 42f07e8c bcfb3fb4 9faac12e b2fee5b6

DPP PKEX Commit/Reveal Request frame from ac:64:91:f4:52:07 to 6e:5e:ce:6e:f3:dd

71876f5a f10f0100 a0000000 a0000000 14000000 02000000 d0000000 6e5ece6e
f3ddac64 91f45207 ffffffff ffff0000 0409506f 9a1a0109 04107800 c23ad1ef
c564aa9a e2f95d9c 2faf67fb 47c038b1 ca1d3327 1186cd38 c3caf63b 14c156ba
366eb37e 70480cc3 43cd3f35 53636d8b 30a04912 a09cee8b 85373c41 3955107e
f5c1bcb3 e15d55ae ac6c6eb8 643ce94d d37c8c1f b978b8d3 803616a5 661dbd54
13668722 6892c300 8497e90d a52eb58b 0838e1d2

Wrapped Data, length 120, value:
c23ad1ef c564aa9a e2f95d9c 2faf67fb 47c038b1 ca1d3327 1186cd38 c3caf63b
14c156ba 366eb37e 70480cc3 43cd3f35 53636d8b 30a04912 a09cee8b 85373c41
3955107e f5c1bcb3 e15d55ae ac6c6eb8 643ce94d d37c8c1f b978b8d3 803616a5
661dbd54 13668722 6892c300 8497e90d a52eb58b 0838e1d2
```

C.7 Responder processes PKEX Commit/Reveal Request frame

```
y:
d98faa24 d7dd3f59 2665d71a 95c862bf d02c4c48 acb0c515 a41cbc6e 929675ea

X.x:
740ab9f0 c173507b 0081b475 b275de6a 3060cf43 4b6a65f0 b0144a1d bf913310

Z.x = (y*X).x:
7415e1c6 8611f044 3cc345d1 36984e48 8c6a26d3 d5482fa6 7e9841a0 3a87c78f

context to create z
MAC-Initiator:
```

```
ac6491f4 5207

MAC-Responder:
6e5ece6e f3dd

M.x:
bcca8e23 e5c05032 ae6051ca 6392f7c4 a4b4f9fe 13e81261 32d070e5 52848176

N.x:
0a91e072 8809bb81 91ea36d0 a1d5602b f36ab670 8fbfd063 e2511e53 3b534020

password:
74686973 69737265 616c6c79 73656372 6574

z:
5271dee9 15cf7b19 08747d8e db839444 2411c518 3ee38b79 ebef399c 08738e0b

y:
d98faa24 d7dd3f59 2665d71a 95c862bf d02c4c48 acb0c515 a41cbc6e 929675ea

A:
0ad58864 754c8126 85ff3a52 a573c1d7 2c72c4eb ed98f391 5622d4df c84a438d

(y*A).x:
31c1b9ab 31d9c2f2 78b35b5c 29d180df eaf76d58 5ede9c0d d91cb661 49db572e

context for u'
MAC-Initiator:
ac6491f4 5207

A.x:
0ad58864 754c8126 85ff3a52 a573c1d7 2c72c4eb ed98f391 5622d4df c84a438d

Y.x:
a9972a94 f143740d f31c7a61 124d01a4 e949d0fd cede6136 9f4c6b09 7aeb18b5

X'.x:
740ab9f0 c173507b 0081b475 b275de6a 3060cf43 4b6a65f0 b0144a1d bf913310

u':
598c3d8d cccea2d4 3259068d 542a9074 42f07e8c bcfb3fb4 9faac12e b2fee5b6

AUTHENTICATED PEER! Bootstrapping key is trusted!

peer's trusted bootstrap key:
4d446b77 45775948 4b6f5a49 7a6a3043 41515949 4b6f5a49 7a6a3044 41516344
49674144 43745749 5a48564d 67536146 2f7a7053 70585042 31797879 784f7674
6d504f52 56694c55 3338684b 5134303d
```

C.8 Responder generates PKEX Commit/Reveal Response frame

```
(b*X).x:
bc5f3128 b0b99707 9a23ead6 3cf502ef 4f752660 22696203 77b79bce 20e03d44

context for v
MAC-Responder:
6e5ece6e f3dd

B.x:
977b7fa3 9779a814 29febb12 e1dc5e20 a7e017c4 bc743709 0e57c966 a2b0e8a3
```

```
X.x:
740ab9f0 c173507b 0081b475 b275de6a 3060cf43 4b6a65f0 b0144a1d bf913310

Y.x:
a9972a94 f143740d f31c7a61 124d01a4 e949d0fd cede6136 9f4c6b09 7aeb18b5

v:
b2833ce2 1ab4e42c 082111a5 dd232334 e48019f6 6b2e274f 521fe2f7 dfa11999

DPP PKEX Commit/Reveal Response frame from 6e:5e:ce:6e:f3:dd to ac:64:91:f4:52:07

71876f5a 336c0100 a0000000 a0000000 14000000 02000000 d0000000 ac6491f4
52076e5e ce6ef3dd ffffffff ffff0000 0409506f 9a1a010a 04107800 00664552
70bcd7c1 73235ca2 86cd867a bb62b1e4 7c8e4de3 c2886316 ffa1bf4f 3e53e617
6770bb9f 4ddebdae 1a4c003b df9c8695 c44493ab 6ade758c 6cf41f0f 7f3f93a6
1b7b6fd4 44b16c5a 2729f346 1b21f6bc d78ac1af e33d0649 9cb5bae0 2d54e949
188e3b9f 8a194f3d d899bd15 6aa54ecf e04b0c84

Wrapped Data, length 120, value:
00664552 70bcd7c1 73235ca2 86cd867a bb62b1e4 7c8e4de3 c2886316 ffa1bf4f
3e53e617 6770bb9f 4ddebdae 1a4c003b df9c8695 c44493ab 6ade758c 6cf41f0f
7f3f93a6 1b7b6fd4 44b16c5a 2729f346 1b21f6bc d78ac1af e33d0649 9cb5bae0
2d54e949 188e3b9f 8a194f3d d899bd15 6aa54ecf e04b0c84
```

C.9 Initiator processes PKEX Commit/Reveal Response frame

```
x:
8365c5ed 93d751be f2d92b41 0dc6adfd 95670889 183fac1b d66759ad 85c3187a

B:
977b7fa3 9779a814 29febb12 e1dc5e20 a7e017c4 bc743709 0e57c966 a2b0e8a3

(x*B).x:
bc5f3128 b0b99707 9a23ead6 3cf502ef 4f752660 22696203 77b79bce 20e03d44

context for v'
MAC-Responder:
6e5ece6e f3dd

B.x:
977b7fa3 9779a814 29febb12 e1dc5e20 a7e017c4 bc743709 0e57c966 a2b0e8a3

X.x:
740ab9f0 c173507b 0081b475 b275de6a 3060cf43 4b6a65f0 b0144a1d bf913310

Y'.x:
a9972a94 f143740d f31c7a61 124d01a4 e949d0fd cede6136 9f4c6b09 7aeb18b5

v':
b2833ce2 1ab4e42c 082111a5 dd232334 e48019f6 6b2e274f 521fe2f7 dfa11999

AUTHENTICATED PEER! Bootstrapping key is trusted!

peer's trusted bootstrap key:
4d446b77 45775948 4b6f5a49 7a6a3043 41515949 4b6f5a49 7a6a3044 41516344
49674143 6c33742f 6f356435 71425170 2f727353 34647865 494b6667 46385338
6444634a 446c664a 5a714b77 364b4d3d


Ini
```