TLS-POK

Proof of Knowledge

draft-friel-tls-eap-dpp

Friel, Harkins

Context

- Wi-Fi alliance Device Provisioning Protocol defines how a device's keypair can be bootstrapped to onboard the device to a Wi-Fi network
- DPP gives the device a guarantee that it is connecting to a network that knows its bootstrap public key, optionally it can provide mutual authentication if device can do bootstrapping too
- The bootstrap public key format is the ASN.1 SEQUENCE SubjectPublicKeyInfo from RFC5280
- Public key in DPP URI may be printed in a QR label, included in a BOM, etc. or it could be obtained through other means such as a cloud service
- We want to reuse the same bootstrap public key to enable a device to securely bootstrap against a wired network using TEAP via a TLS extension, then use TEAP tunnel to securely provision device
- This means that if a device supports both Wi-Fi and wired networks, the same QR, BOM, etc. may be used to establish trust across both Wi-Fi and wired deployments

```
DPP:I:GS-803XL;K:MDkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDIgAC8YIhb0MFjXZzwIS3Ry9c4UAR+VZutTkYnjNLNWWGedE=;;
```

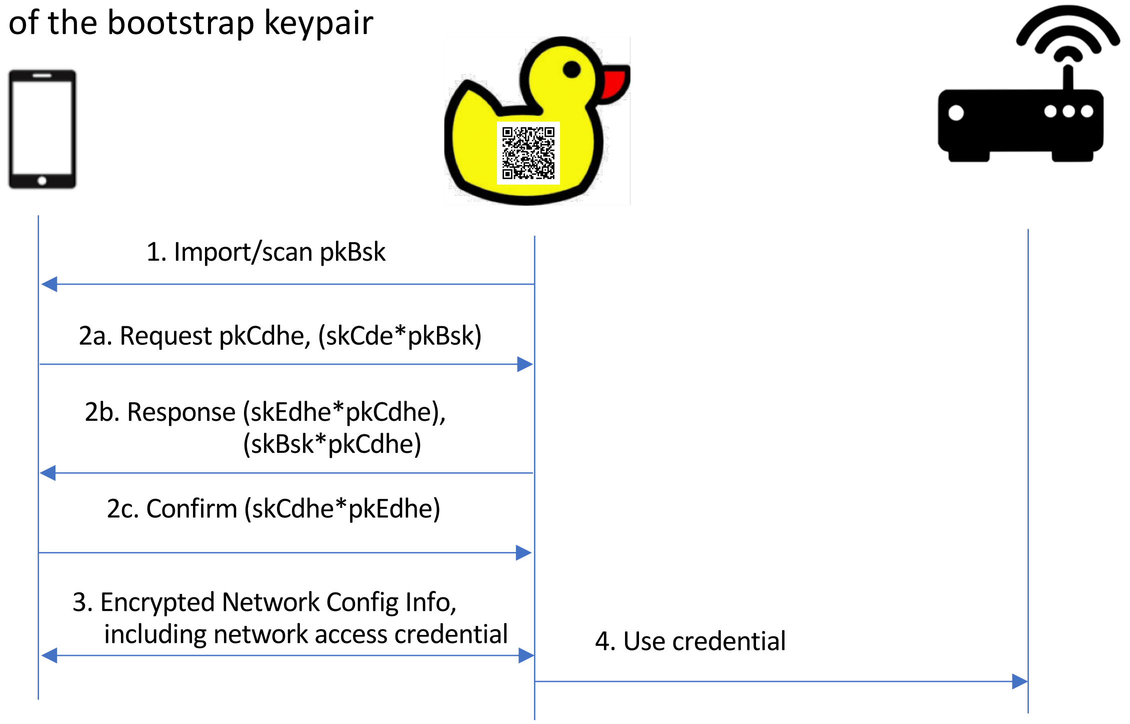
DPP Outline

Mutual auth if both sides bootstrap each other's keys, otherwise Resurrecting Duckling security[1]:

1. Public bootstrap key is provisioned in DPP Configurator
   - Configurator could be a mobile App, or could be be embedded to Wi-Fi AP
   - Many ways to bootstrap– QR code is MTI, NFC, PKEX, cloud, are optional
2. Proof of knowledge via DH using the static bootstrap key and the Configurator ephemeral key
   - Supplicant proves it knows the private key of the bootstrap keypair,
   - Configurator proves it knows the public key of the bootstrap keypair
   - Secure channel established
3. Network information is securely exchanged
4. Supplicant attaches to network

<u>Enrollee bootstrap keypair, DHE keypair</u>
pkBsk, pkEdhe: pubic key
skBsk, skEdhe: private key

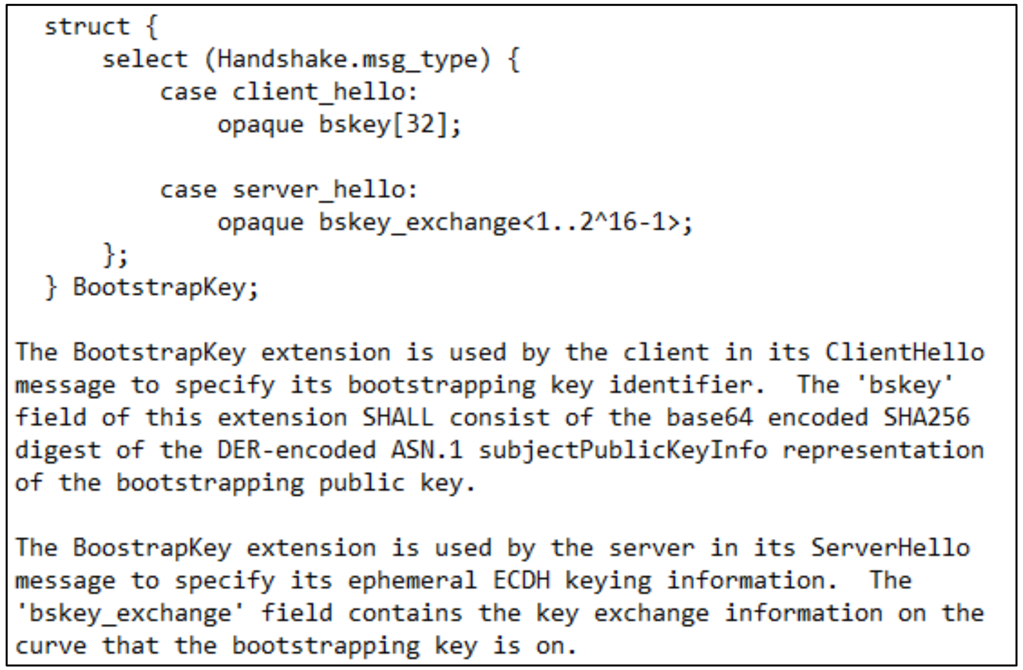
<u>Configurator DHE keypair</u>
pkCdhe: pubic key
skCdhe: private key


1. Import/scan pkBsk
2a. Request pkCdhe, (skCde*pkBsk)
2b. Response (skEdhe*pkCdhe), (skBsk*pkCdhe)
2c. Confirm (skCdhe*pkEdhe)
3. Encrypted Network Config Info, including network access credential
4. Use credential

1 F. Stajano and R. Anderson, The resurrecting duckling: Security issues for ad-hoc wireless networks, in: *International Workshop on Security Protocols* (1999).

Bootstrap key reuse for wired LAN

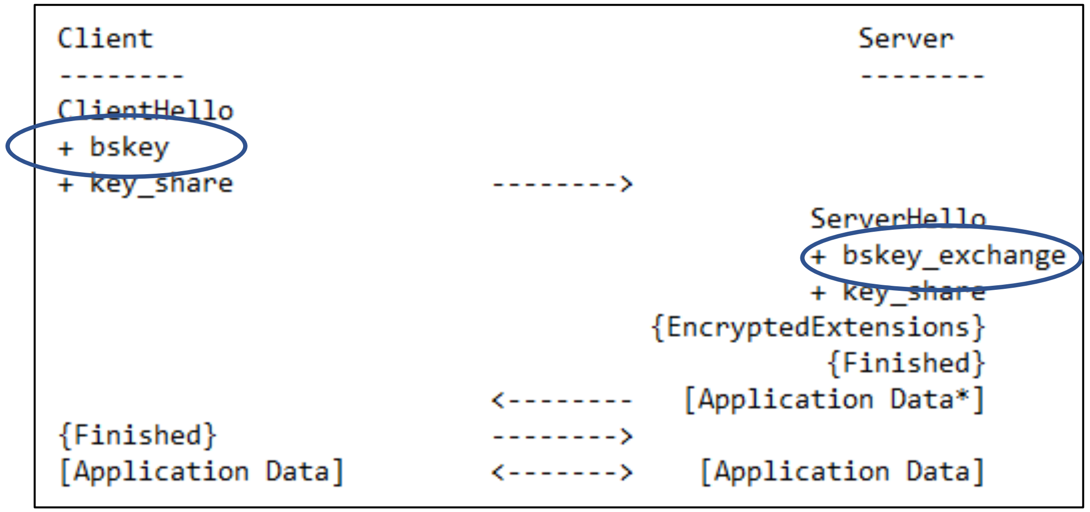
- The pkBsk is scanned into the network and known by the AAA / EAP TLS server
- The device wants the network to prove it knows its pkBsk
- The network wants device to prove it knows private analog of pkBsk
- Can be achieved by exchanging two sets of DH keys in the ClientHello/ServerHello
    1. Standard key_share where both sides generate ephemeral key pairs
    2. Bootstrap extension where client sends its H(pkBsk) instead of pkBsk. Server responds with a second ephemeral key, and uses H(pkBsk) to lookup the actual pkBsk in order to complete its key derivation
- Both DHE calculations are injected into the key schedule



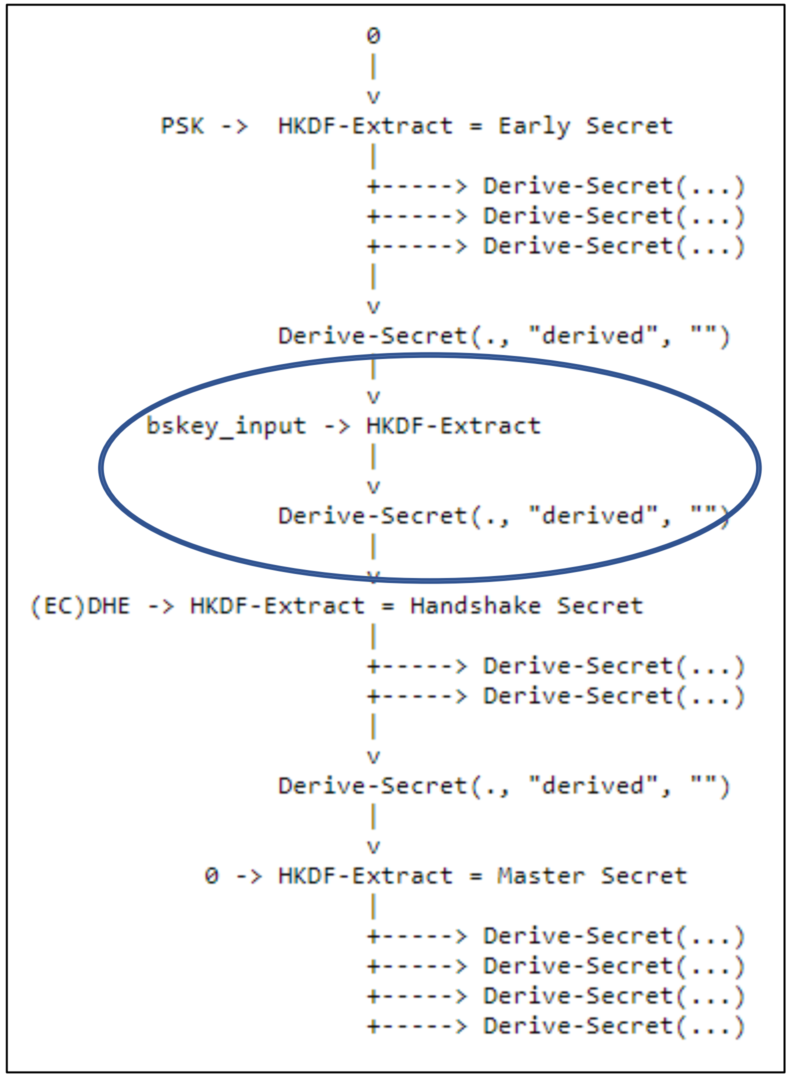
```
  struct {
      select (Handshake.msg_type) {
          case client_hello:
              opaque bskey[32];

          case server_hello:
              opaque bskey_exchange<1..2^16-1>;
      };
  } BootstrapKey;
```

The BootstrapKey extension is used by the client in its ClientHello message to specify its bootstrapping key identifier. The 'bskey' field of this extension SHALL consist of the base64 encoded SHA256 digest of the DER-encoded ASN.1 subjectPublicKeyInfo representation of the bootstrapping public key.

The BoostrapKey extension is used by the server in its ServerHello message to specify its ephemeral ECDH keying information. The 'bskey_exchange' field contains the key exchange information on the curve that the bootstrapping key is on.

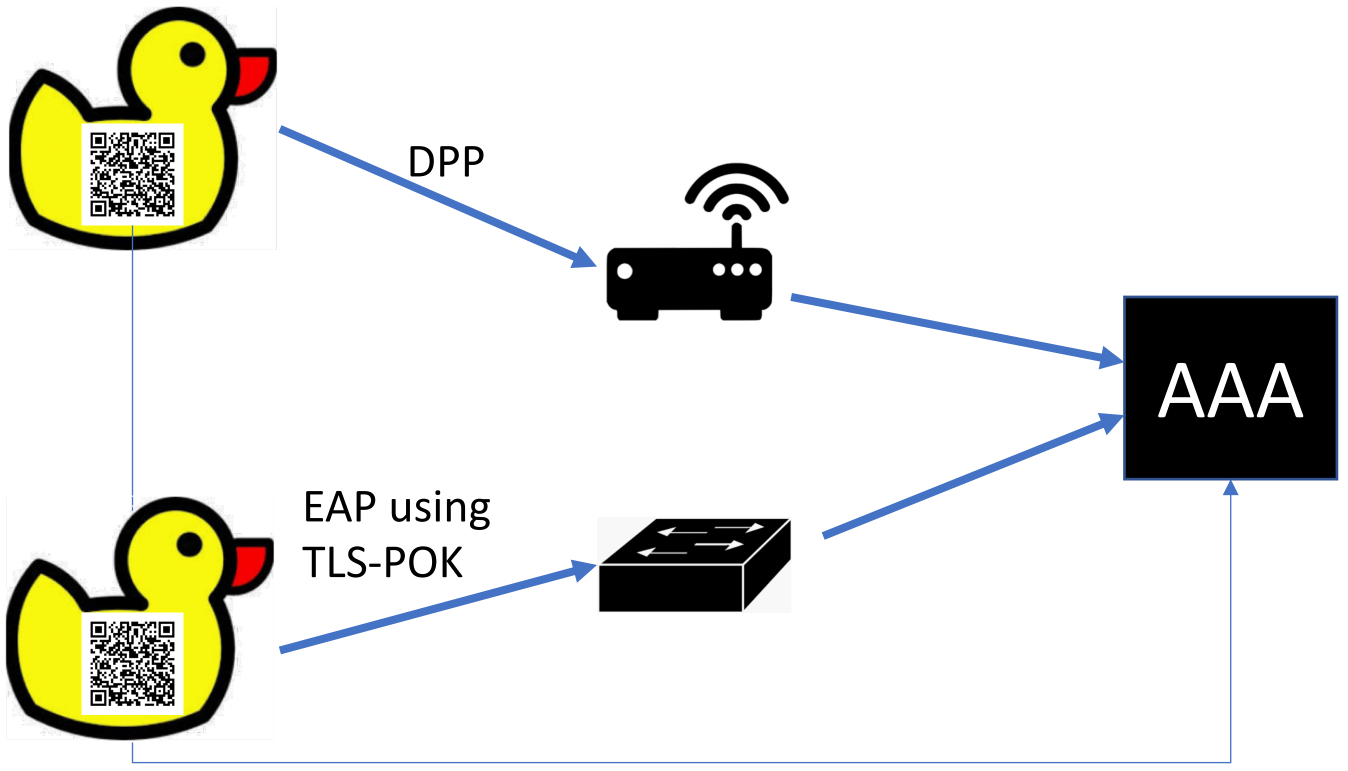


```
Client                                        Server
--------                                      --------
ClientHello
+ bskey
+ key_share              -------->
                                           ServerHello
                                       + bskey_exchange
                                           + key_share
                                 {EncryptedExtensions}
                                            {Finished}
                         <--------  [Application Data*]
{Finished}               -------->
[Application Data]       <------->   [Application Data]
```



```
                 0
                 |
                 v
   PSK ->  HKDF-Extract = Early Secret
                 |
                 +-----> Derive-Secret(...)
                 +-----> Derive-Secret(...)
                 +-----> Derive-Secret(...)
                 |
                 v
           Derive-Secret(., "derived", "")
                 |
                 v
 bskey_input -> HKDF-Extract
                 |
                 v
           Derive-Secret(., "derived", "")
                 |
                 v
(EC)DHE -> HKDF-Extract = Handshake Secret
                 |
                 +-----> Derive-Secret(...)
                 +-----> Derive-Secret(...)
                 |
                 v
           Derive-Secret(., "derived", "")
                 |
                 v
      0 -> HKDF-Extract = Master Secret
                 |
                 +-----> Derive-Secret(...)
                 +-----> Derive-Secret(...)
                 +-----> Derive-Secret(...)
                 +-----> Derive-Secret(...)
```

Everyone is Happy



Scan/import

Security Considerations for Ducklings

- DPP
  - If you know the bootstrap public key, you can claim the device
- TLS-POK
  - If you know the bootstrap public key, you can claim the device

Working Code

- https://github.com/upros/mint/tree/tls-pok