Defining The Problem Space
IoT Ecosystem Challenges

Private PKI everywhere

Establishing trust across private PKI domains is challenging

No technical controls prevent naming collisions across PKI

Discovery API for CA and EE certs oftentimes proprietary

PKI over-consolidation to prevent naming collisions and ease trust challenges

Implied trust in decoupled applications because cert discovery is too difficult
IoT Ecosystem Challenges

Private PKI needs a broadly useful discovery mechanism

Specifically, we need a discovery mechanism which:
- works across private PKI (enabling mTLS authentication)
- works for async applications (object signing/encryption)
- prevents naming collisions
- makes credential rotation easier
- works well for constrained platforms

Create safe bridges between islands of trust
Islands Of Trust

CA bundle makes web PKI work well
   Imagine the web without the browser CA bundle...

IoT typically uses private PKI for client identity
   Web PKI for this use case brings unnecessary cost and complexity

Agreement on IoT roots of trust -> CA cert distribution or consolidation

Multiple CA certs invites naming collision, so consolidation is favored

Entire org on the same trust island, cross-org M2M is difficult
Suboptimal Information Flows
Suboptimal Information Flows (tromboning)
Suboptimal Information Flows

This is what we want:
Assumed Trust In Middleware
Proprietary certificate discovery protocols
Evaluating existing open identity systems

Looking at existing technologies:

- Open standards
- Ease of integration
- Enable existing PKI authn methods to work better
- Give transition to a better state a gradual adoption curve
Lens For Examining Identity Systems

<table>
<thead>
<tr>
<th>Question</th>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>How Accessible?</td>
<td>Entire system</td>
<td>integration</td>
</tr>
<tr>
<td>How widely-recognized?</td>
<td>Name</td>
<td>federation</td>
</tr>
<tr>
<td>How resistant to name collisions?</td>
<td>Name</td>
<td>ambiguity</td>
</tr>
<tr>
<td>How resistant to credential theft?</td>
<td>Proof-of-Possession</td>
<td>resilience</td>
</tr>
<tr>
<td>Level of effort for credential rotation?</td>
<td>Proof-of-Possession</td>
<td>hygiene</td>
</tr>
</tbody>
</table>
## Traditional PKI

<table>
<thead>
<tr>
<th>Question</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>How Accessible? (API)</td>
<td>Proprietary</td>
</tr>
<tr>
<td>How widely-recognized?</td>
<td>Which PKI?</td>
</tr>
<tr>
<td>How resistant to name collisions?</td>
<td>Only within CA</td>
</tr>
<tr>
<td>How resistant to credential theft?</td>
<td>Hardware-supported</td>
</tr>
<tr>
<td>Level of effort for credential rotation?</td>
<td>High</td>
</tr>
</tbody>
</table>
Blockchain and Public Key

<table>
<thead>
<tr>
<th>Question</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>How Accessible? (API)</td>
<td>Standards emerging (DID)</td>
</tr>
<tr>
<td>How widely-recognized?</td>
<td>Which blockchain?</td>
</tr>
<tr>
<td>How resistant to name collisions?</td>
<td>Within same system</td>
</tr>
<tr>
<td>How resistant to credential theft?</td>
<td>Hardware-supported</td>
</tr>
<tr>
<td>Level of effort for credential rotation?</td>
<td>High</td>
</tr>
</tbody>
</table>

Diagram: Name, Proof-Of-Possession, Metadata
DNS+PKI

<table>
<thead>
<tr>
<th>How Accessible? (API)</th>
<th>Already in the OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>How widely-recognized?</td>
<td>Only one DNS</td>
</tr>
<tr>
<td>How resistant to name collisions?</td>
<td>Only one DNS</td>
</tr>
<tr>
<td>How resistant to credential theft?</td>
<td>Hardware-supported</td>
</tr>
<tr>
<td>Level of effort for credential rotation?</td>
<td>Low</td>
</tr>
</tbody>
</table>

15
DANE and LoRaWAN
LoRaWAN Architecture

ED

RG

3G, Ethernet

IS

NS

AS

Backend Network Elements
DNS infrastructure usage in LoRaWAN

- For Over the Air Activation (OTAA)
  - Purpose – ED Onboarding
  - The NS uses the DNS infrastructure to find the JS based on the JS identifier (JoinEUI) in the incoming Join-Request (JR) from the ED

- For Roaming
  - When the ED is roaming in a Visited Network (VN), the Visited NS uses the NetID to identify the home network of the ED
LoRaWAN DNS hierarchy

“lora-alliance.org” -> Lora-Alliance Web Service

“netids.lorawan.net” -> Lora-Alliance Roaming Service

“joineuis.lorawan.net” -> Lora-Alliance OTAA Service
Provisioning JoinEUI and NetID in LoRaWAN

JoinEUI

0x00005E100000002F

NetID

0x60050A

f.2.0.0.0.0.0.0.0.1.e.5.0.0.0.0.joineuis.lorawan.net

60050a.netids.lorawan.net

f.2.0.0.0.0.0.0.0.1.e.5.0.0.0.0.joineuis.lorawan.net. IN  NS example.net

60050a.NETIDS.lorawan.net IN  A 192.0.2.0.1
OTAA flow

ED ➔ JR

RG

NS

DNS

JS

JoinAccept ➔ JoinAccept

Get IP address of the JS

JR + NetID, DevAddr, DLSettings, RxDelay, CFList

JoinAns
Passive roaming flow

ED | RG | fNS | sNS | DNS | JS

- **Passive roaming flow**
- **Get IP address of the JS**
- **Get sNS Identifier of the DevEUI**
- **Get IP address of sNS**
- **IP address of sNS**
- **ProfileReq**
- **ProfileAns**
- **JR**
- **JAccept**
- **JAccept**
- **JAns**
- **JAns**

Packet uplink

Packet downlink
Issues with the current setup (1/2)
Issues with the current setup (2/2)
## Naming conventions, SDOs and Naming Services in IoT

<table>
<thead>
<tr>
<th>Naming Conventions</th>
<th>SDO</th>
<th>Naming Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI (e.g. Domain names)</td>
<td>IETF</td>
<td>DNS</td>
</tr>
<tr>
<td>EPC</td>
<td>GS1</td>
<td>ONS</td>
</tr>
<tr>
<td>OID</td>
<td>ITU and ISO/IEC</td>
<td>ORS</td>
</tr>
<tr>
<td>DOI</td>
<td>ISO</td>
<td>Handle</td>
</tr>
</tbody>
</table>
Goal from DANISH

• Use DNS infrastructure and its security extensions (DNSSEC/DANE) to
  • Provide mutual authentication between the Backend Network elements
  • Evaluate End-to-End secured communication between the ED and the backend network elements using asymmetric keys and DNS infrastructure as PKI
Initial Work Areas
Background: TLS Server Authentication With DANE

DANE Primer:

- “DNS-based Authentication of Named Entities”: RFC 6698, 7671
- Uses DNSSEC to authenticate X.509 certificates and/or public keys

Today, DANE is defined primarily for authenticating the TLS server (in certain applications)

- See also 7672 (DANE for SMTP Transport Security, and 7673 (DANE with DNS SRV records)
_25._tcp.mail.protonmail.ch.7200 IN TLSA 3 1 1 ( 76BB66711DA416433CA890A5B2E5A0533C6006478F7D 10A4469A947ACC8399E1 )

_25._tcp.mail.protonmail.ch. 7200 IN RRSIG TLSA 8 5 1200 ( 20210302081502 20210131081502 6753 protonmail.ch. UVJuhvyEj............... )
port, protocol, domain name

_25._tcp.mail.example.com. IN TLSA ( 3 1 1 d2abde240d7cd3ee6b4b28c54df034b9 7983a1d16e8a410e4561cb106618e971 )
_25._tcp.mail.example.com. IN TLSA (3 1 1 d2abde240d7cd3ee6b4b28c54df034b97983a1d16e8a410e4561cb106618e971)
DANE record specifies the SHA256 hash of the subject public key of the certificate that should match the End-Entity certificate. Authenticated entirely in the DNS (no PKIX involved).
TLS Client Authentication with DANE

- Original drafts developed in mid 2015; refreshed late last year
  - TLS Client Authentication via DANE TLSA Records:
  - TLS Extension to convey DANE Client Identity:

- Target use cases: IOT & SMTP Transport Security
Protocol Summary

- Client has:
  - DNS domain name identity
  - A public/private key pair & a certificate binding the public key to the domain name
  - Corresponding DANE TLSA record published in DNS

- (D)TLS server
  - Sends Certificate Request message in handshake; extracts client identity from presented certificate, constructs TLSA record; queries, and validates DANE TLSA response
Protocol Summary

- New TLS extension for conveying client’s identity
  - For signaling support for DANE TLS client authentication (empty extension if signal only)
  - For conveying client DNS identity when used with TLS raw public key auth (RFC 7250)
  - In TLS 1.3, this extension is carried in the (encrypted) Client Certificate message (in TLS 1.2 it is carried in the first flight Client extension and has no provision for privacy protection)
Client DNS Naming Convention

Draft is not proscriptive, but proposes 2 naming formats that may be generally suitable for many applications.

Format 1: Service specific client identity

_service.[client-domain-name]

e.g. _smtp-client.relay1.example.com

1st label identifies the application service name. The remaining labels are composed of the client domain name. Allows the same client to have distinct authentication credentials for distinct application services.
Client DNS Naming Convention

Format 2: IOT Device Identity

[deviceid]_device.[org-domain-name]

e.g.

a1b2c3._device.subdomain.example.net.

- a1b2c3: device identifier (could be multiple leftmost labels)
- _device: identity grouping label
- subdomain: organizational label (optional)
- example.net: organizational domain
sensor7._device.example.com. IN TLSA ( 3 1 2
  0f8b48ff5fd94117f21b6550aaee89c8
d8adbc3f433c8e587a85a14e54667b25
f4dcd8c4ae6162121ea9166984831b57
b408534451fd1b9702f8de0532ecd03c )
TLS Client e.g. IOT Device

TLS Server e.g. IOT Controller

TLS Handshake Start

Server Certificate; Client Certificate Request

Client Certificate + DANE Indication

Verify client’s certificate against DANE TLSA record in the DNS
**TLS CLIENT**

Key ^ ClientHello
Exch | + key_share*
| + psk_key_exchange_modes*
v + pre_shared_key*

**TLS SERVER**

--------->
ServerHello ^ Key
+ key_share* | Exch
+ pre_shared_key* v
{EncryptedExtensions} ^ Server
{CertificateRequest v Params
  *+DANE Client ID ext}
{Certificate*} ^
{CertificateVerify*} | Auth
{Finished} v

<-------- [Application Data*]

^ {Certificate
  +DANE Client ID ext]}
Auth | {CertificateVerify*}
v {Finished}  -------->

[Verify Client w/ DANE]
[TLS alert on failure ]

[Application Data]  <-------->  [Application Data]
Bridging the Gap

DANE requires DNSSEC, and that’s the endgame, but ..

DNSSEC is not universally deployed (yet). And we’d like to be able to use the DANE, even if DNSSEC has not been deployed in the relevant pieces of the DNS infrastructure.

And interim proposal to (securely) achieve that goal involves the specification of an additional TLSA usage mode that allows the certificate data to be authenticated solely with traditional PKIX.
TLS Mutual Authentication With DANE

Simplify PKI management tasks:
  - Certificate rotation happens via your own DNS.
  - Certificate rotation happens as frequently as desired, TTL is only delay.

Attribution for authenticating peer is straightforward (DNS hierarchy)
TLS Mutual Authentication With DANE
DANE for Certificate Discovery: PKIX-CD

Provide a transitional DANE mode en route to DNSSEC DANE

Two closely-related use cases:

Entity certificate discovery (object security)

Trust anchor certificate discovery (TLS client CA certificates)
DANE for Certificate Discovery: PKIX-CD

New certificate usage mode 4: PKIX-CD

Entity certificate is authenticated by a discoverable CA certificate

CA certificate discovery is authenticated by Web PKI

This is for domains which cannot yet adopt DNSSEC
DANE for Certificate Discovery: PKIX-CD

DANE TLSA record containing an x.509 certificate

    abc._device.vendor.example IN TLSA 4 0 0 ..................

Breaking down the identity name:

    abc:          Device identifier
    _device:      Identity grouping label
    vendor.example:  Organizational domain
DANE for Certificate Discovery: PKIX-CD

Identity name components:
- abc: Device identifier
- _device: Identity grouping label
- device: Identity type (remove underscore)
- vendor.example: Organizational domain
- authorityKeyID: Extracted from entity certificate

CA certificate URI pattern:
https://${ID_TYPE}.${ORG_DOMAIN}/.well-known/ca/${AKI}.pem

https://device.vendor.example/.well-known/ca/AA-BB-CC….pem
DANE for Certificate Discovery: PKIX-CD

Entity certificate >> abc._device.vendor.example

CA certificate >> https://device.vendor.example/.well-known/ca/AA-BB-CC....pem

Discovery protected by Web PKI, in the absence of DNSSEC
PKIX-CD: Caveats

Must ensure name alignment between DNS query and x.509 contents
  Mitigation for same-domain impersonation

Must limit discovered CA certificates to their associated domains
  Mitigation for cross-domain impersonation

Must ensure the uniqueness of distinguishedName in cached CA certificates
  Mitigate TLS 1.3 confusion with trust chains (RFC 8446, sec 4.2.4)
PKIX-CD: Implementation remaining questions

Is the dNSName the best field to use in the certificate for name alignment?

Will underscores in this field cause friction with CAs issuing private PKI?

We could use otherName, but dNSName seems like a better fit.

Should x.509 nameConstraints (RFC5280) be required for CA certificates?
PKIX-CD: Object Security Use Case

Sender’s certificate discoverable in DNS
Message contains the sender’s DNS name
Message is signed by sender’s private key

---

Recipient uses sender’s DNS name to retrieve certificate from DNS
Sender’s signing CA cert discoverable at a URI relative to sender’s DNS name
Discovery of CA certificate protected by traditional Web PKI and TLS
PKIX-CD: Object Security Use Case
PKIX-CD: Object Security Use Case
PKIX-CD: Object Security Use Case
PKIX-CD: Object Security Use Case
PKIX-CD: Trust Anchor Discovery (for mTLS/PKI)

PKIX-CD process requires the discovery of entity and CA certs

Trust chain:
EE-Cert >> Private-CA-Cert >> TLS/WebPKI

Object security (message sign/encrypt) -> Retain EE-Cert
Trust anchor discovery (mTLS/PKI) -> Retain Private-CA-Cert

Note: Trust anchor discovery happens out-of-band, not a part of the TLS handshake.
Network Access Use Cases
PKIX-CD and EAP-TLS use case (CA cert discovery)

Desired behavior:
   Use supplier-provisioned DANE identity for EAP-TLS
   Avoid supplier lock-in

Challenges:
   Onboarding all mfr CA certs is a manual process
   Authz is complicated by different PKI naming conventions
   Prevent cross-CA impersonation

With PKIX-CD and EAP-TLS:
   RADIUS performs authz based on DNS name to ca cert mapping
PKIX-CD and EAP-TLS use case (CA cert discovery)

Supplier responsibility:
- Publish TLSA records for devices, which contain authorityKeyID (AKI).
- Publish authority server (https://device.vendor.example/.well-known/ca/${AKI}.pem)

Implementer responsibility:
- Manage names of authorized identities in RADIUS
- EAP-TLS server CA certificate installation on device

RADIUS automation:
- Periodically update **DANE_ID : AKI** mappings
- Periodic out-of-band CA cert sync

RADIUS AA:
- Perform authn using traditional PKI
- Perform authz based on **DANE_ID : AKI** mapping
PKIX-CD and EAP-TLS (CA cert discovery)
PKIX-CD and EAP-TLS (CA cert discovery)
PKIX-CD and EAP-TLS (CA cert discovery)
PKIX-CD and EAP-TLS (CA cert discovery)
DANE Client ID and EAP-TLS use case

Desired behavior:
   Use supplier-provisioned DANE identity for EAP-TLS
   Avoid supplier lock-in

Challenges:
   Onboarding all mfr CA certs is a manual process
   Authz is complicated by different PKI naming conventions
   Prevent cross-CA impersonation

With DANE Client ID and EAP-TLS:
   RADIUS performs authn/authz based on a list of allowed DNS names
DANE Client ID and EAP-TLS use case

Supplier responsibility:
- Publish TLSA records for devices, which contain authorityKeyID (AKI).
- Publish authority server (https://device.example.com/.well-known/ca/${AKI}.pem)

Implementer responsibility:
- Manage names of authorized identities in RADIUS
- RADIUS Server authentication via:
  - RADIUS server CA certificate installation on device
  - or- tls-dnssec-chain-extension for DANE auth before network access is granted

RADIUS automation:
- Periodically update `DANE_ID : AKI` mappings
- Periodically sync discovered CA certs

RADIUS AA:
- Perform authn/z using DANE traditional PKI
- Perform authz based on `DANE_ID : AKI` mapping
DANE Client ID and EAP-TLS (DANE client auth)
DANE Client ID and EAP-TLS (DANE client auth)
DANE Client ID and EAP-TLS (DANE client auth)
DANE for EAP-TLS Summary

Safely enable mfr-supplied PKI identity for network access

Use with or without DNSSEC

PKIX-CD is a first step into DANE, using WebPKI for trust

DANE for client auth is simpler than PKIX-CD (DNSSEC makes things simpler!)
Ecosystem Interaction: mDNS and DANE
mDNS and DANE Use Case: Dash Cam Retention

Taxi service must retain #days of dash cam footage for insurance compliance

Constraints:
- Cost of onboard storage for all cameras for #days too high
- Cost of cellular bandwidth for video transmission too high
- Local storage at taxi garage/offices preferred

Solution:
- Flush video archives from car’s dash cam to office DVR:
  - Every day at end of shift
  - Using office WiFi
  - Video files are signed by dash cam’s private key
mDNS and DANE Use Case: Dash Cam Retention

1: Connect to garage network using EAP-TLS
mDNS and DANE Use Case: Dash Cam Retention

1: Connect to garage network using EAP-TLS
2: Compose mDNS SRV query from DNS name of DVR:
   DNS_NAME=reorder56_dvr.vendor.example
   MDNS_INSTANCE=reorder56
   SRV_QUERY=reorder56_https_tcp.local
3: MDNS: get SRV for $SRV_QUERY
   HOST=reorder56.local
   PORT=443
4: MDNS: get IP for $HOST
   IP=192.168.1.100
mDNS and DANE Use Case: Dash Cam Retention

1: Connect to garage network using EAP-TLS
2: Compose mDNS SRV query from DNS name of DVR:
   DNS_NAME=recorder56._dvr.vendor.example.
   MDNS_INSTANCE=recorder56
   SRV_QUERY=recorder56._https._tcp.local.
3: MDNS: get SRV for %{SRV_QUERY}:
   HOST=recorder56.local
   PORT=443
4: MDNS: get IP for %{HOST}:
   IP=192.168.1.100
5: Begin TLS handshake for %{IP}:%{PORT}:
   SNs=%{DNS_NAME}
   dane_client_id=car54_auto.taxi.example
mDNS and DANE Use Case: Dash Cam Retention

1: Connect to garage network using EAP-TLS
2: Compose mDNS SRV query from DNS name of DVR:
   DNS_NAME=recorder56_dvr.vendor.example.
   MDNS_INSTANCE=recorder56
   SRV_QUERY=recorder56_https_tcp.local.
3: MDNS: get SRV for ${SRV_QUERY}:
   HOST=recorder56.local
   PORT=443
4: MDNS: get IP for ${HOST}:
   IP=192.168.1.100
5: Begin TLS handshake for ${IP}:${PORT}:
   SN=${DNS_NAME}
   tftp_clientId=car54_auto.taxi.example
6: Perform DANE match against server certificate
   Pattern=${PORT}.${PROTO}.${SNI}
   _443_tcp.recorder56_dvr.vendor.example
mDNS and DANE Use Case: Dash Cam Retention

1: Connect to garage network using EAP-TLS
2: Compose mDNS SRV query from DNS name of DVR:
   DNS_NAME=recorder56._dvr.vendor.example.
   MDNS_INSTANCE=recorder56
   SRV_QUERY=recorder56._https._tcp.local.
3: MDNS: get SRV for $SRV_QUERY
   HOST=recorder56.local
   PORT=443
4: MDNS: get IP for $HOST:
   IP=192.168.1.100
5: Begin TLS handshake for $IP:$PORT
   SNI=$DNS_NAME
dane_client_id=car54._auto.taxis.example
6: Perform DANE match against server certificate
   Pattern: _$PORT._$PROTO.$SNI
     _443._tcp.recorder56._dvr.vendor.example
7: Present client certificate
mDNS and DANE Use Case: Dash Cam Retention

1: Connect to garage network using EAP-TLS
2: Compose mDNS SRV query from DNS name of DVR:
   DNS_NAME=recorder56._dvr.vendor.example.
   MDNS_INSTANCE=recorder56
   SRV_QUERY=recorder56._https._tcp.local.
3: MDNS: get SRV for $SRV_QUERY
   HOST=recorder56.local
   PORT=443
4: MDNS: get IP for $HOST:
   IP=192.168.1.100
5: Begin TLS handshake for $IP:$PORT
   SNI=$[DNS_NAME]_dane
dane_client_id=car54._auto.taxi.example
6: Perform DANE match against server certificate
   Pattern: $[PORT]_[$PROTO]_$[SNI]_443._tcp.recorder56._dvr.vendor.example
7: Present client certificate
8: Compare client certificate against TLSA at car54._auto.taxi.example
mDNS and DANE Use Case: Dash Cam Retention

1: Connect to garage network using EAP-TLS
2: Compose mDNS SRV query from DNS name of DVR:
   DNS_NAME=recorder56._dvr.vendor.example.
   MDNS_INSTANCE=recorder56
   SRV_QUERY=recorder56._https._tcp.local.
3: MDNS: get SRV for $SRV_QUERY
   HOST=recorder56.local
   PORT=443
4: MDNS: get IP for $HOST:
   IP=192.168.1.100
5: Begin TLS handshake for $IP:$PORT
   SN=$DNS_NAME
dane_client=car54._auto.taxi.example
6: Perform DANE match against server certificate
   Pattern: $PORT$_$PROTO$_$SN
   __443__tcp.recorder56._dvr.vendor.example
7: Present client certificate
8: Compare client certificate against TLSA at car54._auto.taxi.example
9: Upload signed dash cam archives to DVR system
Scope of Work
DANISH Core Objectives

DANE for Client Identity:


DANE for Certificate Discovery:

DANISH Peripheral Objectives

HTTP, MQTT, etc:
   HTTP and PROXY protocol header standardization

Email ecosystem:
   STARTTLS, IANA email auth parameters for DANE client ID

Object security:
   JOSE/COSE updates for ‘x5u’ field usage
HTTP and PROXY protocol headers

HTTP proxy headers:
   Need to convey the DANE client name
   Suggest: `X-Forwarded-DANE-Client-Id`

TCP TLVs for PROXY protocol:

   `PP2_TYPE_SSL`: contains `PP2_SUBTYPE_SSL_CN` for cert CN.
   Suggest adding `PP2_SUBTYPE_SSL_DANE` for DANE DNS name
JWS and JWE define x5u field for locating an x509 cert:

https://tools.ietf.org/html/rfc7515#section-4.1.5
https://tools.ietf.org/html/rfc7516#section-4.1.7

Currently:
Only PEM-formatted certificates
Only over HTTPS

Update:
Support DNS URI and DER encoding:

dns:abc._device.example.com?type=TLSA
OR  dns://1.1.1.1/abc._device.example.com?type=TLSA
COSE

COSE defines x5u and x5u-sender fields for OOB discovery of x509 certs

Currently:
Only application/pkix-cert and application/pkcs7-mime media types

Update:
Support for application/dns media type
Eligible for Inclusion
SMIMEA Update

SMIMEA (RFC8162):
    DANE for SMIME certificate discovery
    Naming convention:
        ${LOCAL_PART_SHA}._smimecert.${DOMAIN}
    Requires DNSSEC (https://tools.ietf.org/html/rfc8162#section-6)

Update:
    Allow PKIX-CD for discovery of cert and chain.
    Use PKIX-CD pattern:
    https://smimecert.${DOMAIN}/.well-known/ca/${AKI}.pem
Uses of the TLS DNSSEC Chain Extension


“The DANE Authentication Chain Extension for TLS”

Currently planned to be published through the Independent Submissions Editor (ISE) channel.

But has potential applicability in the IOT space, e.g. for DANE TLS server authentication in Network Access Authentication scenarios where the EAP-TLS client has no initial access to the network to perform DNS queries, ...
DANE for SIP Authentication

Some mention on mailing list, worth exploring further?

Possible outcomes:

- Supplier-issued endpoint identity used for network auth
- Supplier-issued endpoint identity used for SIP client authentication
- Could PKIX-CD make RFC 5922 (domain certs for SIP) easier to implement?
Oauth2 mTLS Update

The client_id is derived from information in the x509 certificate
The jwks_uri points to a JWKS doc (https://tools.ietf.org/html/rfc8705#section-2.2)
JWKS doc contains client certs

Update:
Add dane_uri for indicating TLSA records to be used for mTLS authentication
MLS Protocol


Makes use of a Directory for public key distribution and discovery

Would DNS be a good fit as a directory for MLS?
Appendix

DANE operational guidance:

DANE for Client Identity:

DANE for certificate discovery:
https://github.com/ashdwilson/dane-pkix-cd

Well-Known URIs:

TLS 1.3:

JOSE:
https://datatracker.ietf.org/wg/jose/documents/

COSE:
https://datatracker.ietf.org/wg/cose/documents/

IANA Email Auth Parameters:
https://www.iana.org/assignments/email-auth/email-auth.xhtml

Internet X.509 PKI Certificate and CRL Profiles:

DNSSEC Chain Extension:

Multicast DNS:

EAP-TLS:

Proxy Headers:
https://www.haproxy.com/blog/haproxy/proxy-protocol/

DANE for S/MIME:

SIP Certificate-Based Authentication:

Oauth2: