# Establishing a DANE identity manager with usable security

# Abstract

DNS provides a service to look up servers by name, but this service, by default, lacks authentication and is prone to forgery. Various security protocols and extensions exist, but their complexity is high and, thereby, adoption is increasing only slowly. This thesis focuses on simplifying the management of identities within the DNS-based Authentication of Named Entities (DANE) protocol. A conceptual approach for a DANE identity management application is established, driven by user-centered design, along with a proof-of-concept implementation. This application’s goal is to increase usable security of DNS authentication. By analyzing every aspect of the life cycle of an identity and the certificates associated with it, the conceptual application design is evaluated both in terms of security and usability.

# Table of contents

# 1. Introduction

# 1. Introduction

Problems

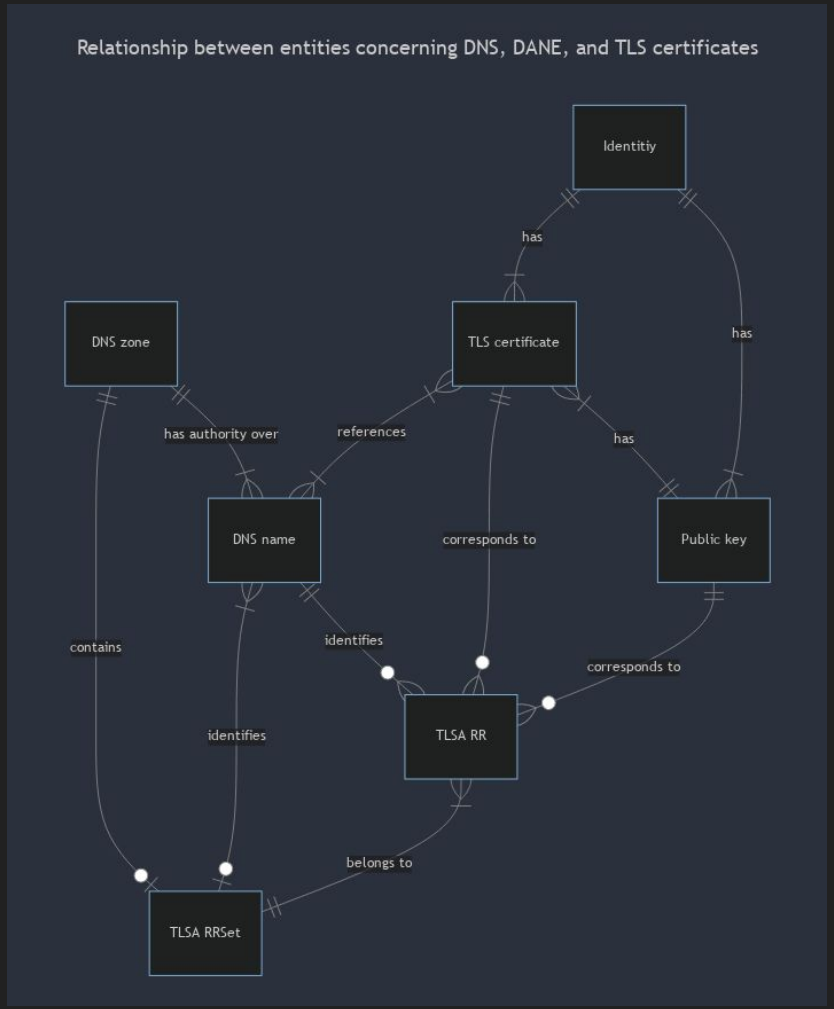
1. Client connects to server, e.g. DNS → IP → TCP → TLS → HTTP, but when TLS handshake is performed, there is no way to know if **identity** of server is correct
2. Manually publishing and updating TLSA RRs is **cumbersome** and prone to **human error**
3. Automatically updating TLSA RRs is not trivial, requires careful **management**, needs to obey different **requirements**, needs to support different **scenarios**

Solutions

1. **DNS-based Authentication of Named Entities (DANE)** allows making TLS certificate and public keys available in DNS using TLSA Resource Record (RR); a client can then simply compare both certificates or public keys → server authentication
2. TLSA RRs can be **automatically published** within a DNS management system; an **intuitive UI** can simplify the process
3. By examining the **life cycle events** of a server identity, the **validity** can be ensured and important **state transitions** can be correctly managed

Need to establish a DANE identity manager that synchronizes TLSA entries with TLS certificates automatically.

# 1. Introduction



# 2. Protocols involved in DANE identity management

# 2.1. Transport Layer Security (TLS) certificates

- Certificate binds **identity** to **public key**—verified by **signature**
- **Signing**: “encrypting” with private key; *self-signed*: signed with one’s *own* private key; **certificate chain**
- Purpose: establishing **secure connection** between server and client (authenticity + confidentiality + integrity)
- CA hierarchy: one **self-signed root certificate** and a few **intermediate certificates** (signed by root cert)—all known by browsers and verified
- Workflow:
  1. Subject generates **public–private key pair**
  2. Subject makes **certificate request** with **identifying information** and **public key**
  3. CA **signs** certificate request by an intermediate certificate, defines validity period
  4. The result is the subject’s **certificate**

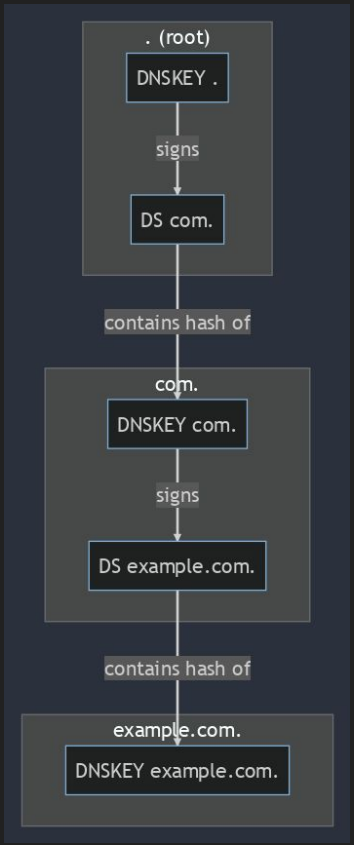
**Standard:**
X.509

**Formats:**
PEM
DER

# 2.2. The Domain Name System (DNS)

- Know a name? Look up its address. “Phone book of the Internet”
- Distributed architecture
- Hierarchy of DNS zones
  - Starts with “`.`”, the root zone
  - DNS name such as “`www.example.com.`” has zone under “`example.com.`”, which has zone under “`com.`”, which has zone under the root zone “`.`”.
- Each zone has set of **resource records (RRs)**, each with a **name**, a **class**, a **type**, and **data**
  - E.g. name “`www.example.com.`” has IPv4 address `93.184.216.34`, because it has RR of class `IN` (internet), type `A`, data `93.184.216.34`.
- Each RR has a wire format and a presentation format
  - In wire format, everything is expressed as compact bits
  - In presentation format, everything is expressed as human-readable text
  - Each type has a number: `A` is number `1`, `AAAA` (for IPv6) has number `28`, `TLSA` has number `52`
- Records for same name, with same class and same type form an **RRSet**

# 2.2.1. Security extensions for the DNS (DNSSEC)




- **RRSIG** record has signature of RRSet of another type and a validity period—must be periodically renewed
- **DNSKEY** records contain public keys; can be ZSK or KSK
  - ZSK is private–public key pair; the private one signs one RRSet
  - KSK is another private–public key pair; the private one signs the DNSKEY RRSet
- **DS** record has hash of ZSK of a DNSKEY
  - Parent zone *owns* this record
  - The DNS name of the record identifies the *child* zone
- **NSEC** identifies next DNS zone in alphabetical order; if no record exists for given DNS query, *this* is used as the DNS response to be signed, thus authenticated

→ Provides security of data exchanged in DNS: authentication + integrity.

DANE is not useful if DNSSEC is missing or signature is invalid.

# 2.2.2. The DANE TLS association (TLSA)

- Class `IN`
- Type `TLSA`
- Three parameters followed by data:
  - `uint8` – Usage
  - `uint8` – Selector
  - `uint8` – Matching type
  - Hex string – Certificate data

} Possible values: “parameter combinations”

**Usage**: PKIX-TA (`0`) PKIX-EE (1) DANE-TA (2) DANE-EE (3)

**Selector**: Cert (`0`) SPKI (1)

**Matching type**: Full (`0`) SHA2-256 (1) SHA2-512 (2)

TLSA records have name like `_443._tcp.www.example.com.`, CNAME RR can provide indirection.

- DANE usages allow certificates without (known) CA → no trust in external CA needed
- PKIX usages require additional checks
- Possible scenarios / considerations:
  - Client could augment certificate chain from server with full certificate from DNS
  - Client might negotiate raw public key instead of certificate
  - DTLS over UDP – Size matters!
  - Algorithm agility
- Validation strategies:
  - Opportunistic Security
  - Certificate Transparency – not recommended for DANE usages

# 2.2.2. The DANE TLS association (TLSA)

**RRSet** – TLSA RRSet can be grouped by parameter combination

- Every group must be sufficient to authenticate server
- Client tries all groups until it finds supported parameter combination where authentication succeeds

**Validity of RRSet** – Recommendations

- SHA2-512 is not universally supported; if it is used, so must SHA2-256
- `3 1 1` is the most recommended parameter combination; `2 0 1` is also recommended
- `X 0 0` and `2 1 X` should be avoided

**Client and server requirements**

- For `3 1 X` or `3 0 0`, server *may* omit certificate; for `3 0 0`, client might not support this
- For `3 X X`, client *must ignore* validity period of certificate, but *must honor* it for `2 X X` and `1 X X`
- If `2 X X` exists, but not all of them are `2 0 0`, then server *must* provide full certificate chain
- Server *must* include TA in certificate chain for `2 1 1` or `2 1 2` and *should* do so for `2 1 0`
- etc.

# 2.2.3. Other DANE associations: SSHFP, IPSECKEY, SMIMEA, OPENPGPKEY

| | | |
|---|---|---|
| IPSECKEY | RFC 4025 | IPsec keying material |
| SSHFP | RFC 4255 | SSH fingerprints |
| OPENPGPKEY | RFC 7929 | OpenPGP public keys |
| SMIMEA | RFC 8162 | S/MIME certificate associations |

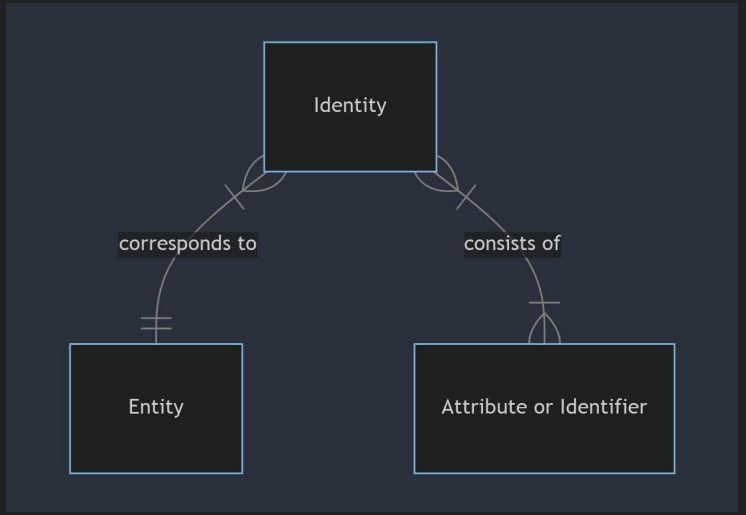
# 2.3. Related problems solved by similar protocols

RFC 7817 TLS server identity check for email-related protocols

RFC 8484 DNS over HTTPS

# 3. The standards aspect of identity management

# 3.1. Terminology




- An **identity** is a set of descriptions matching an **entity** (user or object) which identifies said entity – *“website” (web service, server, etc.)*
- **Identification** is indication of an identity
- An **identifier** is an external description, meaning that the entity can be reached by it from outside – *“hostname” (DNS name, address)*
- **Authentication** is the process of verifying an identity – *happens via a public key infrastructure*
- A **certificate** contains an identity and a public key and binds them to each other

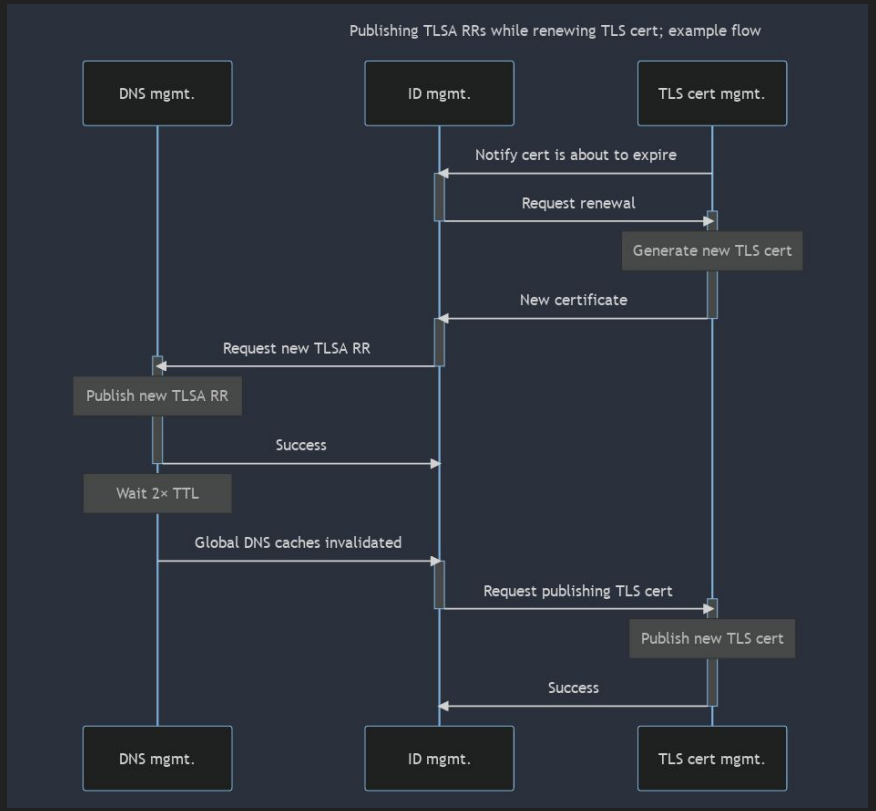
Perspective:

- (Client’s) *reference* identity (of server) – with “reference identifier”
- (Server’s) *presented* identity (of server)

“Authenticator” vs. “Credential”: RFC 4949: “***Identifier credential**: A data object that is a portable representation of the association between an identifier and a unit of authentication information, and that can be presented for use in verifying an identity claimed by an entity that attempts to access a system. Example: X.509 public-key certificate.*”; NIST uses “authenticator” like “token”—a public key can be an authenticator.

“Identity proofing” vs. “Authentication”: NIST 800-63-3: “***Identity proofing** establishes that a subject is who they claim to be. Digital **authentication** establishes that a subject attempting to access a digital service is in control of one or more valid authenticators associated with that subject’s digital identity.*”, “*The process used to verify a subject’s association with their real world identity is called **identity proofing**.*”

# 3.2. Identity life cycle and Identity management

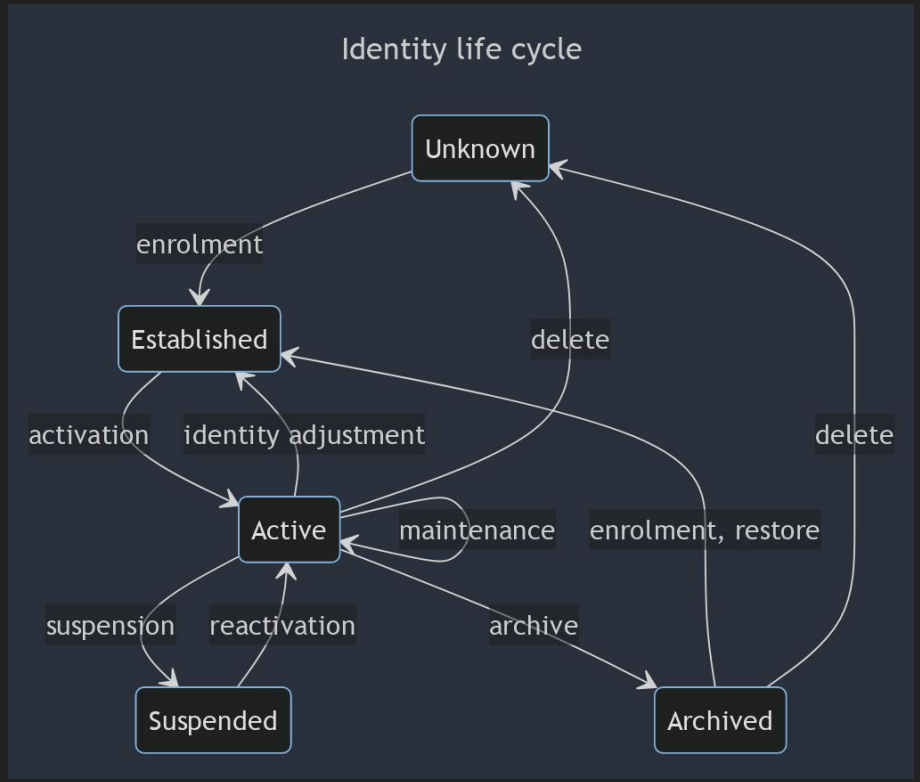



**Important transitions**

- Certificate renewal
- Key rollover
- Switching to a different usage or matching type
  - 1 1 1 → 3 1 1
  - 3 X X → 2 X X
  - X X 1 → X X 2
- Removing a TLSA matching type

TLS certificate revocation has complications, but removing a TLSA RR is easy.

# 3.2. Identity life cycle and Identity management




ISO/IEC 24760

TODO: `Life-cycle analysis.md`

# 3.3. Information security

ISO 27001

- Man-in-the-middle attacks
- Lack of trust in CA
- Availability concerns

# 3.4. Usability and accessibility

**Usability**

- …

**Accessibility**

- WAI-ARIA

# 4. User-centered design implementation using web technologies

# 4.1. host, dig

DNS queries

```
host -t TLSA _443._tcp.www.huque.com.

_443._tcp.www.huque.com has TLSA record 3 1 1 244B484761B1CC2A390DB3F1031C1D512AF3C9C14DF47854E43E3C2917EF671C
_443._tcp.www.huque.com has TLSA record 3 1 1 33F5B0C36FBDE25D8B148623DEB22A4CD5D1A806EE1259F48DD9ABDF36EE8A63
_443._tcp.www.huque.com has TLSA record 3 1 1 DE4369CF0866A1E7626D73DB36DBFC4B74097C3C70489A2D3351B6E75E99583A
_443._tcp.www.huque.com has TLSA record 3 1 1 7634E0EED7DFCA738733ED09C8BED00D4C5F4A3E15F76BC9E0B6E57FDBB9163D
_443._tcp.www.huque.com has TLSA record 3 1 1 EE7041856CC8AD23789D39462A7D11C0B7633C6142ADE119BC9B1BEF708F7D0A
_443._tcp.www.huque.com has TLSA record 3 1 1 6C85CC093C31221CBFF9E61CFF5E9CA14BFEB0F9BBC341A7695290275D813CF4
_443._tcp.www.huque.com has TLSA record 3 1 1 CD98C1661E982DA43E414E9EEB7A3A545DEEC8859F5282E28CA3CBB382DDBE3E
_443._tcp.www.huque.com has TLSA record 3 1 1 A67924AFCD895B9661C4C5D67A83215F60B7D0E1DA8A30B67EEC6BD623A5B57C
```

# 4.2. nmap

TLS requests

```
nmap -vv -p 443 --script ssl-cert www.huque.com

PORT   STATE SERVICE REASON
443/tcp open  https   syn-ack
| ssl-cert: Subject: commonName=www.huque.com
| Subject Alternative Name: DNS:www.huque.com
| Issuer: commonName=R3/organizationName=Let's Encrypt/countryName=US
| Public Key type: rsa
| Public Key bits: 2048
| Signature Algorithm: sha256WithRSAEncryption
| Not valid before: 2023-03-05T11:22:30
| Not valid after:  2023-06-03T11:22:29
| MD5:   550bd8b39d412e7dd8e65f6d60202325
| SHA-1: f30a8a4d1732ebb03a6401c0132e36a9517d8354
| -----BEGIN CERTIFICATE-----
| MIIFITCCBAmgAwIBAgISAyLz82FZiGus5shPARQpGnuAMA0GCSqGSIb3DQEBCwUA
| MDIxCzAJBgNVBAYTAlVTMRYwFAYDVQQKEw1MZXQncyBFbmNyeXB0MQswCQYDVQQD
| EwJSMzAeFw0yMzAzMDUxMTIyMzBaFw0yMzA2MDMxMTIyMjlaMBgxFjAUBgNVBAMT
| DXd3dy5odXF1ZS5jb20wggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDI
| /5Bm1QHenwmyiG8CWUTdrI/09ZC3A9HjlGCWaHx/LJRN4X/VsMSEY3GoTn8UYFTO
…
```

# 4.3. OpenSSL

X.509 certificate management and TLS client

- `openssl genpkey`
- `openssl rsa`
- `openssl pkeyutl`
- `openssl req`
- `openssl verify`
- `openssl x509 -pubkey`
- `openssl s_client`

# 4.4. Node.js backend and API

Integrate the aforementioned tools.

Future proofing:

- Matching type: crypto agility
- Selector: OpenSSL to parse X.509 files and encode DER
- Usage: potentially very versatile, difficult to predict

# 4.5. Web application and user interface

DNS Management and
Identity Management

TODO:
`Conceptual design of a DANE identity manager.md`

# 5. Conclusion and future work